Altti Helläkoski

Linux-yhteisön merkitys tuotekehitysprojektille

Elektroniikan, tietoliikenteen ja automaation tiedekunta

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 10.5.2010.

Työn valvoja:

Prof. Heikki Saikkonen

Työn ohjaaja:

Prof. Heikki Saikkonen

Aalto-yliopisto
Teknillinen korkeakoulu

AALTO YLIOPISTO
TEKNILLINEN KORKEAKOULU

DIPLOMITYÖ
TIIVISTELMÄ

Tekijä: Altti Helläkoski

Työn nimi: Linux-yhteisön merkitys tuotekehitysprojektille

Päivämäärä: 10.5.2010 **Kieli**: Suomi **Sivumäärä**: 7 + 52 + 36

Tietotekniikan laitos

Elektroniikan, tietoliikenteen ja automaation tiedekunta

Professuuri: Ohjelmistotekniikka T-106

Valvoja: Professori Heikki Saikkonen

Ohjaaja: professori Heikki Saikkonen

Vuosittain maailmanlaajuisesti myydään yli miljardi matkapuhelinta. Kilpailu mobiililaitteiden valmistajien kesken on kireää ja laitteiden tuotekehityssykli on nopea. Tämä asettaa yhä tiukempia aikataulullisia ja laadullisia vaatimuksia laitteiden ohjelmistonkehitykselle. Avoimen koodin käyttö on nähty yhtenä ratkaisuna kehitystyön nopeuttamiseksi ja ohjelmistojen laadun parantamiseksi.

Tässä työssä tutkittiin niitä hyötyjä, mitä avoimeen lähdekoodiin perustuvaa käyttöjärjestelmää eli kerneliä käyttävä tuoteprojekti saa toimiessaan tiiviissä yhteistyössä Linux-yhteisön kanssa. Tutkimuksen kohteena oli Nokia N900-laitteen ohjelmistoprojekti. Kyseisen tuotteen kernel-projekti sopi hyvin tutkimuksen kohteeksi, sillä se tapahtui kiinteässä yhteistyössä Linux-yhteisön kanssa, ja työssä tarvittava materiaali on julkisesti saatavilla.

Työn tuloksina todetaan kaupallisen tuoteprojektin saavan erittäin merkittävän määrän valmista ohjelmakoodia ja merkittävän määrän ohjelmakoodin katselmointiapua avoimen lähdekoodin projektilta. Näiden lisäksi todetaan avoimen ohjelmakoodin kehitystavan tarjoavan käyttökelpoisen foorumin teknisten ongelmien ratkaisemiseen ja teknisen tiedon levittämiseen.

Avainsanat: Linux, kernel, kernel-yhteisö, koodin kontribuointi, katselmointi

AALTO UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS
ABSTRACT

Author: Altti Helläkoski

Name of the thesis: Importance of the Linux-community for a product development project

Date: 10.5.2010 **Language:** Finnish **Pages:** 7 + 52 + 36

Department of Computer Science and Engineering

Faculty of Electronics, Communications and Automation

Professorship: Software Technoilogy Code T-106

Supervisor: Prof. Heikki Saikkonen

Instructor: Prof. Heikki Saikkonen

More than one billion mobile phones are sold each year globally. On top of that there is hundreds of millions other mobile devices like music players and navigators. Competition between mobile device manufacturers is tough, and a gap between device generations has shorten all the time. This has produced some rigid requirements to product development projects. Those requirements emphasize time to market and comprehensive set of features. Usage of open source software has been seen as a possible solution to fulfill these requirements.

This thesis studies the advantages that a product kernel development project can expect to gain while working in close co-operation with Linux kernel development community. Nokia N900 kernel development project was one of the projects that worked in a close co-operation with Linux kernel developers. For that reason the project was chosen to be analyzed in this thesis. Development followed the practices of open source development. This makes it possible to analyze the kernel development by examining available mail archives.

As a result of the research it was found that the project got a large amount of program code and review work from the Linux kernel community. Additionally, Linux kernel development mailing lists were a useful way to solve technical problems and to spread detailed technical information.

Keywords: Linux, kernel, kernel community, code contribution, code review

Esipuhe

Tämän diplomityön valvojana ja ohjaajana toimi professori Heikki Saikkonen. Esitän hänelle kiitokseni kärsivällisyydestä ja työtä koskevista neuvoista ja kommenteista. Työn rakenteen tarkistamisesta ja ohjeistuksesta haluan lausua kiitokseni diplomi-insinööri Taina Hyppölälle.

Haluan kiittää Linux-spesialisti Topi Kanervaa ja diplomi-insinööri Tony Lindgreniä lukuisista Linuxin kernel-kehitykseen liittyvistä tiedoista. Haluan myös kiittää vaimoani varatuomari Heidi Dahlqvistiä kieliasua ja oikeinkirjoitusta koskevista kommenteista. Kiitän kaikkia työtovereitani ja Linux-yhteisön asiantuntijoita, jotka ovat olleet suureksi avuksi tämän työn valmistumisessa.

Espoo 10.5.2010

Altti Helläkoski

Sisällysluettelo

Symbolit ja lyhenteet

Ajuri	Fyysisen laitteen ohjaamisesta vastaava ohjelmiston osa.
BT	Bluetooth, langattoman tiedonsiirron standardi.
DSP	Digitaalinen signaaliprosessori.
GIT	Linuxin kernelin kehityksessä käytettävä versionhallintajärjestelmä.
GPL	GNU Public Lisence, avoimen ohjelmistokoodin lisenssi.
GPLv2	Linuxin kernelin ohjelmakoodin suojaava versio GPL-lisenssistä.
GPU	Graphical Processing Unit, grafiikkaprosessori.
IETF	Internet Engineering Task Force, verkkovalmistajien avoin yhteisö, joka kehittää alan standardeja
IPv4	Internet Protocol version 4. RFC 791:n määrittelemä liikennöintiprotokolla
IPv6	Internet Protocol version 6. RFC 2460:n määrittelemä liikennöintiprotokolla, tarjoaa isomman osoiteavaruuden kuin Ipv4.
Kernel	Käyttöjärjestelmän ydin.
Linux	Avoimeen ohjelmistokoodiin perustuva käyttöjärjestelmä.
N900	Nokian Linux-käyttöjärjestelmään pohjautuva puhelimen sisältävä mobiililaite.
NAND	Luku/kirjoitusmuisti, joka säilyttää tilansa vaikka käyttöjännitteet on katkaistu.
Nokia	Nokia Oyj, mobiililaitteita valmistava yritys.
OMAP	Mobiililaitteissa käytetty prosessoriperhe.
OMAP3	Tässä työssä tutkitussa laitteessa käytetty prosessori.
Patch	Ohjelmakoodin muutoksen sisältävä erotiedosto.
PYTHON	Tulkattava, oliopohjainen ohjelmointikieli.
RAM	Random access memory, luku- ja kirjoitusmuisti.
RFC	Request For Comments, yleisnimi IETF:n tuottamalle verkkostandardille, myös standardin kuvaava dokumentti.
ROM	Read only memory, lukumuisti.

RFC 2822	Sähköpostin formaatin määrittelevä IETF:n standardi.
SHA-1	Secure Hash Algorithm 1, RFC 3174:ssä määritelty kryptografinen tiivistemenetelmä, jonka avulla digitaalisesta tiedostosta voidaan muodostaa yksilöllinen tunniste.
TI	Texas Instruments Inc. Prosessorivalmistaja
UNIX	Käyttöjärjestelmä
USB	Universal Serial Bus, sarjaliikenne- ja liitinstandardi.
WLAN	Wireless Local Area Network, langattoman tiedonsiirron standardi.

1. Johdanto

Tässä diplomityössä tutkittiin niitä hyötyjä, joita yrityksen kernel-kehitysprojekti saa toimiessaan tuotekehitysaikana tiiviissä yhteistyössä Linux-yhteisön kanssa. Tutkimushypoteesina oli, että projekti hyötyy yhteistyöstä saadessaan Linux-yhteisöltä katselmointiapua ja osan tarvittavasta ohjelmakoodista.

Huolimatta siitä, että Linuxin käyttöä yrityksissä ja sen merkitystä yrityksille, on tutkittu runsaasti, ei kaupallisen tuotteen kernel-projektin avoimuudesta saavutettavia hyötyjä käsittelevää tutkimusta ole tämän kirjoittajan tiedossa. Tämä diplomityö pyrkii osaltaan täyttämään tuon aukon.

Linux-käyttöjärjestelmä oli alun perin yhden henkilön aloittama harrastusprojekti[1]. Se sai kuitenkin nopeasti taakseen laajenevan kehittäjäjoukon ja käyttäjäkunnan yliopistojen opiskelijoiden ja aktiivisten ohjelmankehittäjien parista. Merkittävä osoitus käyttöjärjestelmän soveltuvuudesta kaupallisiin tuotteisiin saatiin vuonna 1995, jolloin julkaistiin Linuxilla toteutettu Apache verkkopalvelinohjelmisto[2]. Apachen julkaisusta lähtien Linuxin käyttö yritysmaailmassa on kasvanut jatkuvasti. Linux on tullut tunnetuksi luotettavana ja edullisena käyttöjärjestelmänä[3]. Linux-käyttöjärjestelmä skaalautuu järeistä keskustietokoneista aina kannettaviin tietokoneisiin asti. Viimeisen viiden vuoden aikana sen käyttö myös sulautetuissa järjestelmissä on kasvanut. Linux löytyy yhä useamman matkapuhelimen ja musiikkisoittimen sisältä.

Linux-käyttöjärjestelmä koostuu laitteen resursseista vastaavasta kernelistä, sekä sen yhteydessä toimivista kirjastoista, työkaluista ja kernelin ulkopuolella suoritettavista ns. käyttäjän ohjelmista. Tapa jolla Linuxin kerneliä kehittävä avoimen lähdekoodin projekti toimii, eroaa yritysten perinteisesti käyttämistä ohjelmakoodin kehitystavoista. Linuxin kehitys tapahtuu inkrementaalisesti, avoimia postituslistoja käyttäen, niin että kuka tahansa on vapaa kehittämään ohjelmakoodia tai kommentoimaan koodiin tehtyjä muutoksia[4].

Työn aineistona käytettiin kaupallisen mobiililaitteen kernelkehityksestä julkisesti saatavissa olevia tietoja. Kaikkien tahojen kyseiselle projektille tuottama Linuxin kernel-ohjelmakoodi on avoimesti jokaisen tutkittavana ja kommentoitavana.

Työssä käytettiin esimerkkinä Nokian N900-laitteen ohjelmistoprojektia. Valitun tuotteen kernel-projekti toimi kiinteässä yhteistyössä Linux-yhteisön kanssa. Tämän kaupallisen projektin kernel-kehitys tapahtui alusta alkaen niin, että kehittäjät

julkaisivat kaiken kernel-koodin mahdollisimman nopeasti sen valmistumisen jälkeen. Julkaisu tapahtui avoimen koodin kehitysmallin mukaisesti, lähettämällä valmis ohjelmakoodi katselmoitavaksi julkiselle postituslistalle. Lähetettyä ohjelmakoodia parannetaan, kunnes Linuxin kernel-yhteisö toteaa sen vastaavan vaatimuksia ja hyväksyy sen osaksi julkista Linux-kerneliä[4]. N900-projektissa käytettyjä postituslistoja olivat linux-omap (prosessoriarkkitehtuuri), linux-usb (USB-liityntä), linux-mmc (MMC[6]- ja SD[7]-muistikortit) ja linux-alsa (audio-ajurit).

Suurin osa yleisistä ja ns. laiteajureihin kohdistuneista muutoksista on julkaistu lähettämällä ne linux-omap-postituslistalle. Työssä kerättiin tiedot kaikista tarkastelun alla olevan ohjelmakoodin saamista katselmointikommenteista ja korjausehdotuksista. Näiden lisäksi tilastoitiin projektin kehittämään laitteeseen mukaan otettu, ulkopuolisten tahojen projektin aikana kehittämä ohjelmakoodi.

Postituslistojen analyyseissa käytettiin Python-kielistä ohjelmaa. Python valittiin sen nopean ja helpon kehitettävyyden, sekä laajojen postituslistojen käsittelyyn sopivien kirjastofunktioiden vuoksi. Osa tiedoista jouduttiin analysoimaan tutkimalla postituslistoja käsin.

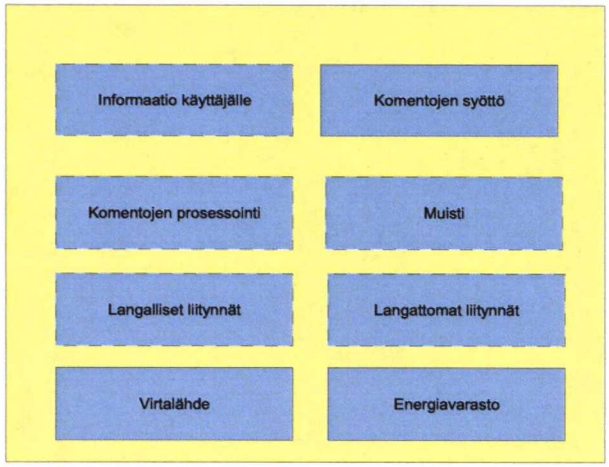
Kappaleessa 2 käydään läpi mobiililaitteiden kehitykseen, Linux-käyttöjärjestelmän historiaan, ja sen kerneliin ja kernelin kehitysmalliin liittyviä, tämän tutkimuksen kannalta keskeisiä asioita. Kernelin kehitysprosessi, ja sen vaikutukset ohjelmakoodia kehittävän ohjelmoijan kannalta, on esitetty yksityiskohtaisemmin. Lopuksi käydään läpi aiheesta aiemmin tehtyjä tutkimuksia ja esitetään lyhyt yhteenveto niiden tuloksista. Diplomityön kappaleessa 3 kuvataan käytetty tutkimusmenetelmä, käytetyt analyysimenetelmät ja ohjelmat. Kappaleessa 4 esitetään tutkimuksen yksityiskohtaiset tulokset. Kappaleessa 5 esitetään yhteenveto tutkimuksen tuloksista. Kappaleessa 6 pohditaan tulosten yleispätevyyttä, arvioidaan mahdollisia virhelähteitä ja tehdään ehdotus aiheen jatkotutkimukseksi.

2. Tutkimuksen taustaa

Tässä kappaleessa määritellään se, mitä mobiililaitteella työn yhteydessä tarkoitetaan. Sen jälkeen tutustutaan niihin syihin, jotka mahdollistivat Linuxin yleistymisen mobiililaitteissa. Seuraavaksi käydään läpi Linuxin rakennetta, Linux-kernelin kehitystapaa ja avoimen ohjelmiston lisenssejä. Lopuksi esitellään muita Linuxin hyötyjä analysoineita tutkimuksia.

2.1 Mobiililaite

Mobiililaite on henkilökohtainen, kannettava, omalla teholähteellä varustettu, tiettyyn tarkoitukseen tehty laite. Nykyisin mobiililaitteella tarkoitetaan yleensä matkapuhelinta, musiikkisoitinta, navigaattoria tai pienikokoista kannettavaa tietokonetta. Nykyaikaiset älypuhelimet ovat esimerkkejä laitteista, joissa on yhdistetty useampia näistä toiminnoista. Kuvassa 1 on esitetty mobiililaitteen periaatekuva. Kiinteällä viivalla esitetyt osiot löytyvät kaikista laitteista, katkoviivalla esitetyt ovat valinnaisia ja riippuvat laitteen toiminnasta.



Kuva 1. Mobiililaitteen lohkokaavio.

Yksinkertainen mobiililaite, esimerkiksi Applen iPod Shuffle-soitin (kuva 2), sisältää vain komentojen syötön, virtalähteen, energiavaraston ja massamuistin, sekä langallisen liittymän kuulokkeita varten. Toista äärilaitaa edustaa tässä työssä tutkitussa projektissa

Kuva 2. Yksinkertainen sulautetun ohjelmiston laite, Apple iPod Shuffle.

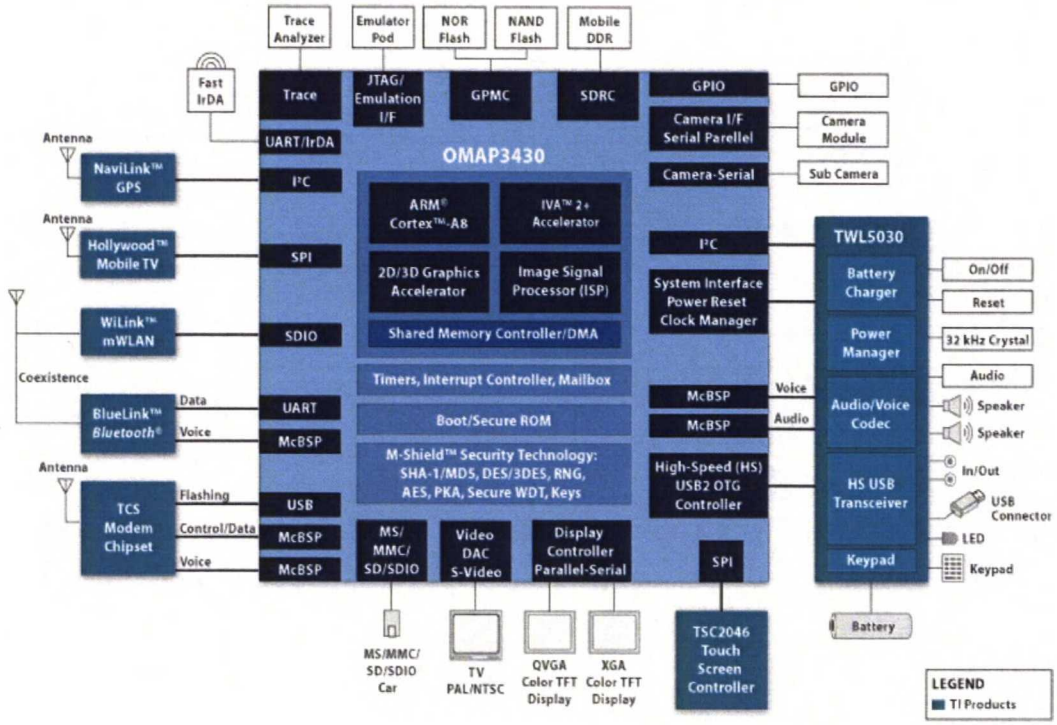
kehitetty Nokia N900-laitte (kuva 3), jossa on kaikki kuvassa 1 esitetyt toiminnalliset lohkot. Käytännössä mobiililaite on useimmiten pienikokoinen tietokone, joka sisältää prosessorin, muistit, teholähteen, sekä tiedon syöttöön ja esittämiseen tarvittavat laitteet. Prosessori on yleensä mobiililaitteita varten suunniteltu, vähän tehoa kuluttava malli.

Tällaisen vähän tehoa kuluttavan prosessorin yhteyteen on integroitu suurin osa tarvittavista oheispiireistä. Osa näistä oheispiireistä on itsessään prosessoreita, jotka mahdollistavat järjestelmän monipuolisen ja joustavan käytön.

Kuva 3. Nokia N900 mobiililaite.

Tässä työssä tarkastellun projektin tuote käyttää TI:n valmistamaa OMAP3-prosessoria,

jossa pääprosessorina on 32-bittinen ARM Cortex-A8[9]. Piirille on pääprosessorin lisäksi integroitu mukaan raskaiden multimediasovellusten vaatima signaaliprosessori, DSP , ja erillinen grafiikkaprosessori, GPU. Näiden prosessointiyksiköiden lisäksi OMAP3 tarjoaa kattavan joukon sarja- ja rinnakkaismuotoisia liityntöjä, joilla voidaan toteuttaa tarvittavat langalliset ja langattomat liitynnät. Kuvassa 4 on lohkokaavio OMAP3-prosessorista. Tarkka kuvaus prosessorin- ja sen oheispiirien toiminnasta on luettavissa vapaasti saatavissa olevasta OMAP3 Technical Reference-dokumentista[10].

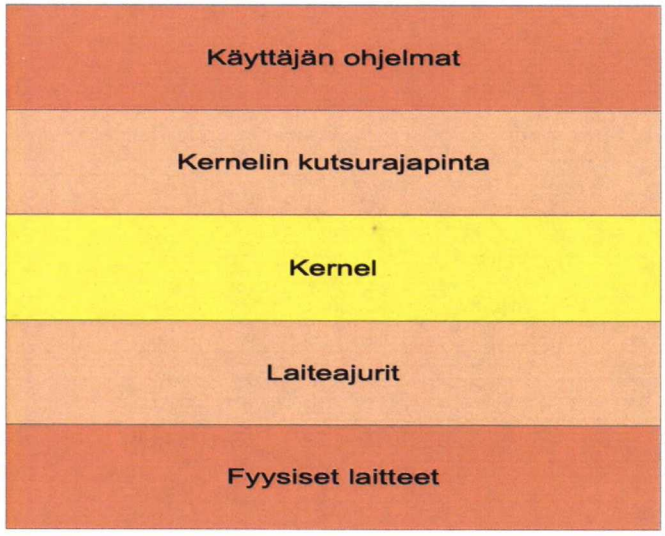


Kuva 4. OMAP3-prosessori liityntämahdollisuuksineen (alkuperäinen kuva OMAP3 Technical Reference Manual).

2.2 Mobiililaitteen ohjelmistot yleensä

Fyysisen laitteiston toimintaa ohjaava ohjelmisto on laaja. Ohjausohjelman kirjoittaminen vaatii prosessoriympäristön, ohjelmointikielten ja liikennöintiprotokollien hallitsemista. Kuvassa 5 on esitetty yleinen

tietokonejärjestelmän rakennemalli, joka kuvaa miten ylempi kerros aina rakentuu alemman varaan. Myös mobiililaitteet noudattavat tätä mallia. Loppukäyttäjän ns. sovellusohjelmat kuten esimerkiksi tekstieditori tai musiikkisoitin, sijaitsevat ylinnä. Sovelluksen kannalta kernel hallitsee kaikkia järjestelmän resursseja. Niihin sovellusohjelma pääsee käsiksi kernelin tarjoamien palvelujen avulla, joita se kutsuu kutsurajapinnan kautta. Palveluja ovat mm. muistin varaaminen, levyllä tai muulla massamedialla olevien tiedostojen käyttö, kommunikointi muiden sovellusten ja laitteiden kanssa jne.



Kuva 5. Ohjelmiston loogiset tasot ja niiden suhde laitteistoon.

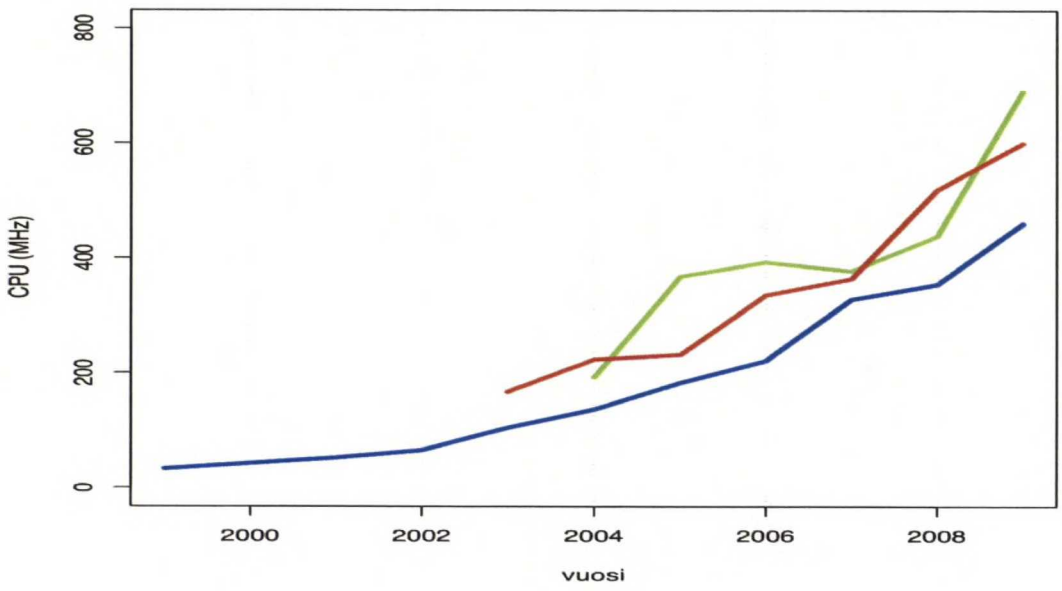
Kernel myös jakaa tiheällä taajuudella prosessorin suoritusaikaa samanakkisesti suorituksessa olevien ohjelmien kesken. Kun mobiililaitteen käyttäjä painaa laitteen näppäintä, vaikkapa kirjainta "a", siirtyy tieto siitä laiteajurin ja kernelin kautta sovellusohjelmalle. Mikäli sovellusohjelma haluaa tulostaa painetun kirjaimen laitteen näytölle, on sen kutsuttava sopivaa kernel-ohjelmaa. Näin menetellen pyritään suojaamaan muut ohjelmat yhden ohjelman virheellisen toiminnan vaikutuksilta. Linux-käyttöjärjestelmän ohjelmistoarkkitehtuuri on kuvattu yleisellä tasolla tämän dokumentin kappaleessa 2.5.

Jokainen prosessoripiirin looginen toimilaite tarvitsee laiteajurin. Laiteajuri on ohjelma, joka piilottaa elektroniikan toiminnan ja tarjoaa ajurin käyttäjälle loogisen rajapinnan laitteen resursseihin. Laiteajurin tehtävänä on laitteen käynnistyksen yhteydessä alustaa

laitteen sähköiset komponentit toimimaan halutulla tavalla. Laiteajuri vastaa myös hallitsemansa laitteen ajonaikaisesta toiminnasta. Mobiililaitteissa energian säästäminen on ensiarvoisen tärkeää. Mikäli laitteella ei ole aktiivista tehtävää käynnissä, tallettaa laiteajuri hallitsemansa laitteen tilan muistiin ja ilmoittaa laitteen resursseista vastaavalle ohjelmalle olevansa lepotilassa. Näin keskitetty resurssienhallinta voi tehonkulutuksen minimoimiseksi sammuttaa käyttöjännitteet ja kellot kyseiseltä elektroniikkaosiolta. Laiteajurin ohjelmisto joutuu myös useissa tapauksissa korjaamaan tai piilottamaan elektroniikan virheellisen tai epätoivotun toiminnan.

2.3 Mobiililaitteiden teknologian kehitys

Viimeisen 10 vuoden aikana on ollut kaksi merkittävää tekijää, jotka ovat mahdollistaneet mobiililaitteiden, kuten musiikkisoittimien ja matkapuhelimien, vallankumouksen. Ensinnäkin elektroniikan integrointiaste on kasvanut. Tämä on mahdollistanut yhä monipuolisempien laitteiden valmistamisen riittävän pienikokoiseksi. Toisaalta elektroniikan tehonkulutus on pienentynyt prosessoreiden käyttöjännitteiden laskiessa. 1980-luvulla tyypillinen sulautetun

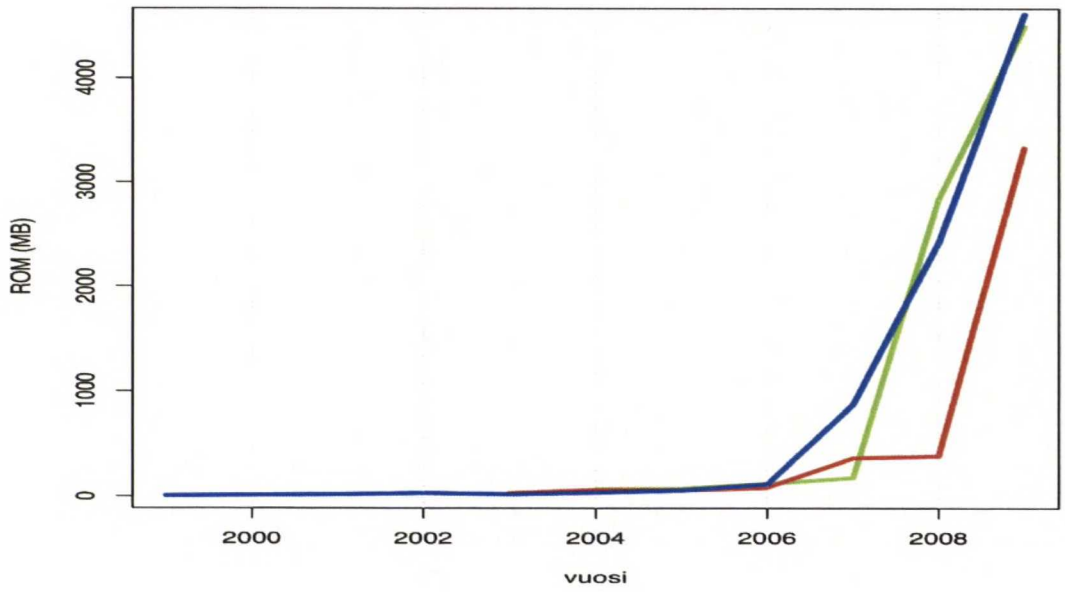


Kuva 6. Prosessoreiden kellotaajuuden keskiarvojen muutos matkapuhelimissa. Sinisellä värillä on merkattu Nokian, punaisella Motorolan ja vihreällä Samsungin älypuhelimet. Kuvan tiedot koottu liitteestä 1.

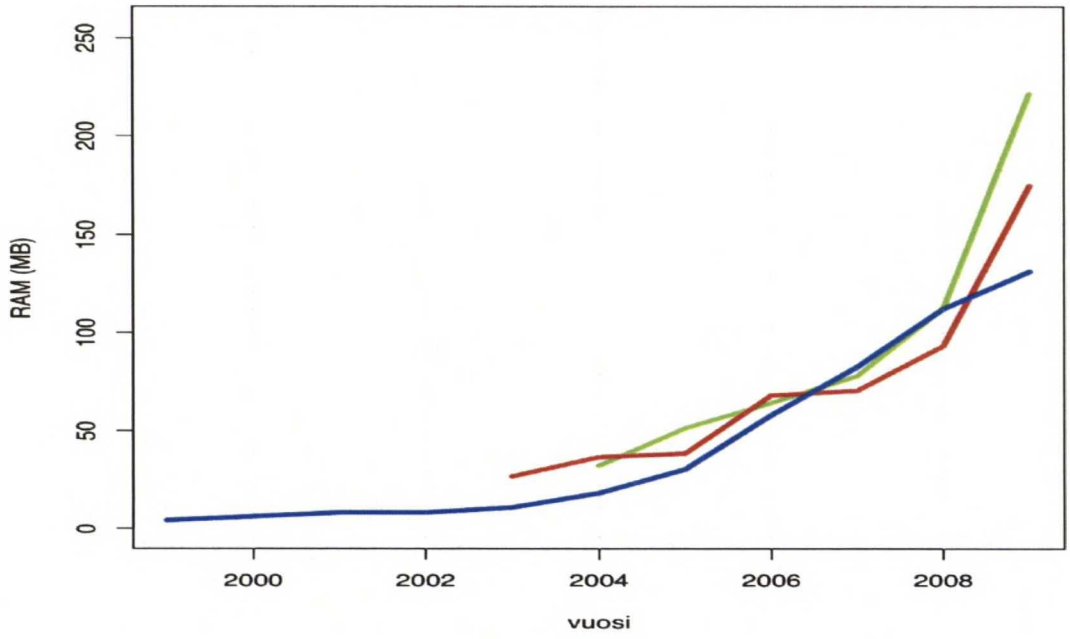
järjestelmän prosessorin käyttöjännite oli 5V[8], tässä työssä tutkitun projektin käyttämä OMAP3-prosessorialusta toimii 0.8-1.3V jännitteellä[10]. Mobiililaitteiden muisti toimii nykyisin 1.8V jännitteellä[11]. Piiri- ja prosessointiteknologian kehitys on mahdollistanut yhä useamman transistorin pakkaamisen yhä pienempään tilaan.

Matkapuhelimissa käytettyjen prosessoreiden suorituskyky on kasvanut viimeisen 10 vuoden aikana yli kymmenkertaiseksi (kuva 6). Samaan aikaan laitteissa käytettävissä olevan muistin määrä on mahdollistanut yhä monipuolisempien ja enemmän muistia vaativien sovellusten toteuttamisen. Kuvassa 6 on esitetty kolmen eri valmistajan matkapuhelimissa käytettyjen prosessorien kellotaajuuksien keskiarvon kasvu.

Ensimmäinen puhelin jossa oli tietokonetyyppisiä ominaisuuksia tarjolla, Nokia 9110i, käytti Intel 486-yhteensopivaa AMD Elan-Prosessoria, jonka kellotaajuus oli 33 Mhz[12]. Tässä työssä tutkittu Nokian N900 käyttää ARM Cortex-A8-prosessoriydintä, jonka kellotaajuus on 600 MHz[9]. Vuonna 2009 julkaistun Samsung "Buckingham"-puhelimen ARM 1176-prosessorin kellotaajuus on 800 Mhz[12].



Kuva 7, ROM-muistin määrän keskiarvon kehitys älypuhelimissa. Sinisellä värillä on merkattu Nokian, punaisella Motorolan ja vihreällä Samsungin älypuhelimet. Kuvan tiedot koottu liitteestä 1.



Kuva 8, RAM-muistin määrän keskiarvon kehitys älypuhelimissa. Sinisellä värillä on merkattu Nokian, punaisella Motorolan ja vihreällä Samsungin älypuhelimet. Kuvan tiedot koottu liitteestä 1.

Samaan aikaan kellotaajuuksien kasvun kanssa on mobiililaitteiden muistin määrä kasvanut merkittävästi (kuva 7 ja 8). Väliaikaiseen talletukseen tarkoitettu luku/kirjoitusmuisti, RAM-muisti, on kasvanut kymmenkertaiseksi. Tämä on mahdollistanut usean sovellusohjelman rinnakkaisen suorittamisen. Tiedon pidempiaikaiseen tallettamiseen tarkoitetun ROM-muistin määrän kasvu monikymmenkertaiseksi on mahdollistanut musiikin, valokuvien ja karttatiedon tallettamisen mobiililaitteiden muistiin. Tiedot on koottu Nokian, Samsungin ja Motorolan julkaisemien älypuhelinten tiedoista vuosilta 1999-2009. Täydellinen lista kyseisistä laitteista löytyy liitteestä 1. Laitteisiin kiinteästi integroitujen muistien lisäksi on useisiin laitteisiin mahdollista lisätä muistia, käyttämällä erillisillä muistikortteja. Nämä kortit voivat olla kapasiteetiltaan useiden gigatavujen kokoisia[5].

2.4 Mobiililaitteen tuotekehitysprosessin haasteet

Ohjelmiston lopputuotteeseen kohdistuvista kustannuksista merkittävä osa muodostuu

lisensoiduista kokonaisuuksista. Tyypillisesti lisenssimaksu maksetaan jokaisesta valmistetusta tuotteesta. Lisenssimaksujen merkitys tuotteen kannattavuudelle on sitä suurempi, mitä isommista valmistusmääristä on kyse. Tämä seikka on saanut useat mobiililaitteiden valmistajat valitsemaan uusiin tuotteisiinsa käyttöjärjestelmän, jonka käyttö ei edellytä lisenssimaksujen maksamista. Tässä työssä analysoidussa projektissa käytettiin GPLv2-lisenssiä[14] käyttävää Linux-käyttöjärjestelmää. Linuxin käyttö ei edellytä lisenssimaksujen maksamista. Varsinaisen käyttöjärjestelmän lisäksi tuotteeseen joudutaan useissa tapauksissa lisensoimaan mediakoodekkeja näihin erikoistuneilta yrityksiltä. Näitä koodekkeja käytetään video- ja audiodatan reaaliaikaiseen purkamiseen ja pakkaamiseen. Nämä koodekkiohjelmat ajetaan joko varsinaisella prosessorilla tai erityisellä kiihdytinpiirillä. Mediakoodekit voidaan useissa tapauksissa ostaa koko laitetuotannon kattavilla kertamaksuilla, eikä niiden tuotteelle aikaansaama kustannus silloin riipu suoraan myytävien laitteiden määrästä.

Ohjelmistoon liittyvät kustannukset koostuvat ostettujen ohjelmistojen lisenssimaksuista, muun ohjelmiston tuottamiseen liittyvistä kuluista ja näistä koostuvan kokonaisuuden toiminnan testauksesta ja laadun varmistamisesta. Ohjelmiston tapauksessa kustannukset jakautuvat suoraan myytyjen tuotteiden kesken. Vain lisensoitavat osat saattavat aiheuttaa kiinteän laitekohtaisen kustannuksen. Perinteisesti mobiililaitteen ohjelmisto on joko tehty kokonaan itse, tai se on lisensioitu näiden valmistamiseen erikoistuneelta yritykseltä. Esimerkki tällaisesta, lisensioitavia käyttöjärjestelmästä, on Microsoft Corporationin Windows CE[13]. Vaikka ohjelmisto lisensoitaisiin kokonaan kolmannelta osapuolelta, vaatii sen räätälöinti, integrointi ja testaus yleensä merkittävän tuotekehityspanoksen.

Tärkein ohjelmiston valmistamiseen liittyvä päätös on käyttöjärjestelmän valinta. Yrityksen strategia saattaa määritellä mitä käyttöjärjestelmää sen valmistamissa tuotteissa käytetään. Käyttämällä kaikissa tuotteissa samaa käyttöjärjestelmää saadaan kasvatettua ohjelmiston uudelleenkäytettävyyttä. Lisäksi saavutetaan etua sillä, että ohjelmistosuunnittelijoiden kompetenssi kasvaa ja ohjelmiston laatu paranee tuotteesta toiseen. Kiristyvä kilpailu on kuitenkin saanut aikaan sen, että yhä enenevässä määrin pyritään valitsemaan uuteen tuotteeseen sillä hetkellä vaatimuksiin parhaiten soveltuva ja kokonaisedullisin käyttöjärjestelmä.

Mobiililaitteiden suosio ja niiden myynnistä saatavat voitot ovat houkutelleet alalle vuosien mittaan yhä kasvavan määrän maailmanluokan valmistajia. Kovenevasta kilpailusta johtuen laitesukupolvet elävät yhä lyhyemmän aikaa, niitä valmistavien yritysten tuotekehitysorganisaatioille asetetaan yhä kasvavia vaatimuksia. Nämä vaatimukset

koskevat niin tuotekehityksen nopeutta kuin laitteisiin sisällytettävien ominaisuuksienkin. Kehittyneimpiä mobiililaitteita voidaan verrata jo kannettaviin yleiskäyttöisiin tietokoneisiin.

Valmistettavan tuotteen myynnille arvioidaan aikaikkuna, jolloin sitä tarjotaan loppukäyttäjien ostettavaksi. Aikaikkuna riippuu teknologian kehityksestä, muodista ja trendeistä, sekä arvioista siitä, millaisia malleja kilpailevat yritykset tuovat markkinoille. Tuotesyklin kiihtymisen vuoksi useimpien laitteiden myynti-ikkuna on kaventunut ja vaihtelee tuotteesta riippuen muutamasta kuukaudesta muutamaan vuoteen. Tuotteen virallinen julkaisu ja myyntiin tulo voivat vaikuttaa ratkaisevasti myytyjen laitteiden määrään. Myöhässä markkinoille tullut laite jää kilpailijan modernimman tuotteen varjoon. Suunnitellusta aikaikkunasta myöhästyminen johtaa useassa tapauksessa koko tuoteprojektin keskeyttämiseen ja tuotekehityspanoksen siirtämiseen jonkin toisen tuotteen valmistamiseen.

Projektin vaatimukset pyritään tekemään niin huolellisesti kuin mahdollista. Tästä huolimatta ne yleensä muuttuvat projektin kuluessa. Muutokset johtuvat markkinoiden jatkuvasta elämisestä; muoti muuttuu, kilpailijat tuovat uusia tuotteita myyntiin ja teknologiassa tapahtuu kehitystä, josta ei ole ollut tietoa alkuperäisten vaatimusten asettamisen aikana. Myös tuotteen kehityksen aikana ilmenneet resurssi-, tekniset tai aikatauluongelmat aiheuttavat usein muutoksia laitteen vaatimuksiin.
Tuotekehitys alkaa valmistettavan tuotteen määrittelystä. Tässä tuote segmentoidaan tiettyyn hinta- ja suorituskykyluokkaan. Tämä kategorisointi antaa pohjan uuden tuotteen vaatimuksille. Ensimmäisenä valitaan riittävän suorituskykyinen prosessoriympäristö. Mobiililaitteissa energiatehokkuus on suorituskyvyn lisäksi hyvin tärkeä tekijä. Tämän takia ei yleensä voida käyttää samoja teknisiä ratkaisuja kuin kiinteästi sähköverkkoon kytketyissä laitteissa. Mobiililaitteissa käytetään tyypillisesti pitkälle jalostettua prosessoriteknologiaa, jossa samalle fyysiselle piirille on integroitu itse suorittimen lisäksi mahdollisimman monet liityntä- ja kiihdytinpiirit. Tällä ratkaisulla säästetään niin tilaa kuin energiaakin, ja tuotanto tulee yksinkertaisemmaksi komponenttien määrän vähetessä. Tyypillinen esimerkki mobiilisuorittimista on Texas Instrumentsin OMAP-tuoteperhe, jonka OMAP3-prosessoria käytetään tässä työssä tutkitussa projektissa. Suorittimen valinta määrittelee pitkälle sen, mitä muita oheispiirejä laitteen sisälle on mahdutettava, sekä sen millaiseen energiankulutukseen on varauduttava. Nämä asettavat puolestaan vaatimukset valmistettavan tuotteen energialähteelle ja kotelolle.

Ohjelmistolle asetetaan varsinaisen laitteen toiminnallisuuden lisäksi vaatimuksia

elektroniikan suunnittelun ja valmistuksen tukemiseksi. Sen halutaan toimivan apuna itse laitteen suunnittelussa. Ohjelmiston tulisi toimia apuna elektroniikan kytkentöjen ja toiminnallisuuden testaamisessa. Tämän lisäksi sen tulee auttaa antennien ja puhekoodekkien virittämisessä, sekä laitteen tuottamien radiotaajuisten häiriöiden analysoinnissa. Nykyaikaisessa mobiililaitteissa voi olla omat antennit esimerkiksi 3G-, WLAN- ja BlueTooth-käyttöä varten. Näiden tehot on säädettävä toisilleen ja ulkopuolisille laitteille aiheutettavien häiriöiden minimoimiseksi. Puhekoodekkien toiminnan optimointi vaatii niiden signaalikäsittelyn säätämisen ja virittämisen. Laitteen kotelon ja piirilevyn mekaaniset mitat ja materiaalit tekevät antennien optimoinnista haastavan tehtävän. Tuotannossa ohjelmiston on pystyttävä toteamaan tuotannon toimivuus, valmistetusta laitteesta pitää nopeasti ja luotettavasti pystyä testaamaan kaikkien komponenttien virheetön toiminta. Tämän lisäksi laitteen ohjelmiston on sopeuduttava yrityksen valmistusmenetelmiin, jotka tyypillisesti aiheuttavat vaatimuksia ohjelmiston toiminnalle.

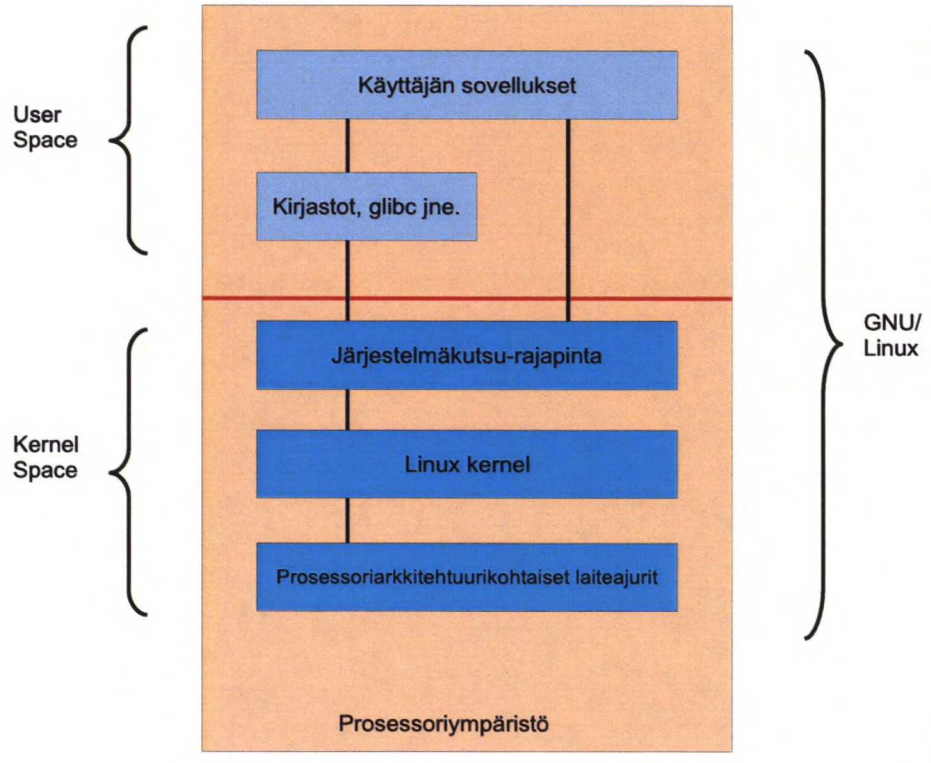
2.5 Linuxin kernel

Yleensä Linuxilla tarkoitetaan niin käyttöjärjestelmän kernelosaa, kuin myös käyttöliittymää ja sovellusohjelmia. Tässä työssä keskitytään Linuxin kerneliin, viitattaessa Linuxiin tarkoitetaan vain sen kerneliä. Mikäli tässä työssä viitataan koko käyttöjärjestelmään, tehdään se kirjoittamalla GNU/Linux.

Linux on hajautetusti kehitetty, vapaasti käytettävissä oleva UNIX-tyyppinen käyttöjärjestelmä. Linux-projekti sai alkunsa Helsingin Yliopiston opiskelijan, Linus Torvaldsin harrastusprojektista, jonka tarkoituksena oli tuottaa luotettava ja ilmainen käyttöjärjestelmä Intel x86-pohjaisille tietokoneille[1]. Alun perin Linux oli vain opetustarkoitukseen kehitetyn Minixin toiminnan parantamiseksi suunniteltu kernel. Torvalds kertoi hankkeestaan hyvin aikaisessa vaiheessa Internetin keskustelupalstoilla. Tämä sai aikaan sen, että suuri joukko harrastajia eri maista osallistui Linuxin syntymästä asti sen kernelin kehittämiseen. Hyvin nopeasti Minix korvattiin GNU-ohjelmilla[15]. Tästä alkanut Linuxin kehitys on vuosien kuluessa tuottanut yhden menestyksekkäimmistä ja laajakäyttöisimmistä käyttöjärjestelmistä. Alussa Linux tuki vain Intelin x86-prosessoriperhettä, mutta nykyisin käyttöjärjestelmä tukee käytännössä kaikkia tunnettuja prosessoriarkkitehtuureja. Tämä on mahdollistanut Linuxin käytön myös mobiililaitteiden käyttöjärjestelmänä. Linux jakautuu arkkitehtuurisesti kahteen osaan, kerneltilaan (engl. kernel space) ja käyttäjätilaan (engl. user space). Kerneltilassa ajetuilla ohjelmilla on pääsy kaikkiin laitteiston resursseihin, käyttäjätilassa pääsyä on

rajoitettu ja se on tehtävä kernelkutsujen avulla.

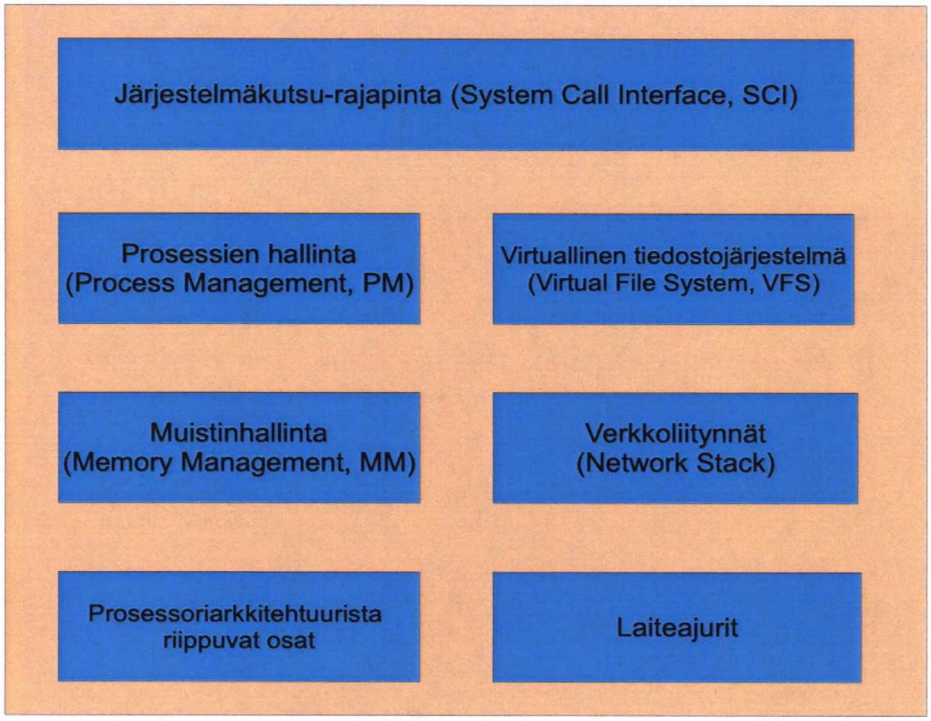
Kuvassa 9 on esitetty yksinkertaistettu malli Linuxin arkkitehtuurista. Ylimpänä kuvassa ovat käyttäjän ohjelmat, kuten webselain tai tekstieditori. Alapuolella on kernel-osio. Jokaisella käyttäjän ohjelmalla on oma suojattu muistialueensa, kernel-moodissa kaikki ohjelmat näkevät koko laitteen muistin, niin kernelin kuin käyttäjän puolellakin olevan. Linuxin kernel jaetaan kolmeen loogiseen osa-alueeseen. Ylimpänä on järjestelmäkutsujen rajapinta, joka toteuttaa yksinkertaisia komentoja, kuten kirjoittamista tai lukemista tiedostosta. Keskellä on varsinainen arkkitehtuureista riippumaton kernelohjelmisto. Alimpana on adaptaatiokerros eri prosessoriarkkitehtuurien sovittamiseksi käyttöjärjestelmään. Adaptiokerroksessa on laitekohtaiset ajurit, joiden tehtävänä on alustaa fyysiset piirit toimimaan halutulla tavalla ja välittää tietoa muiden ohjelmien ja laitteiden välillä.



Kuva 9. Linuxin arkkitehtuurimalli.

Linuxin kernel on monoliittinen ja se tukee moniajoa ja sivutusta. Linuxin yhteydessä kernelillä tarkoitetaan kaikkea kernelmoodissa ajettavaa koodia. Se kattaa käyttöjärjestelmän peruspalvelujen lisäksi myös laiteajurit ja tiedostojärjestelmät.

Kernelin nykyisestä laajuudesta kertoo sen lähdekoodin koko. Vuonna 1994 julkaistu kernelversio 1.0.0 sisälsi 176250 riviä ohjelmakoodia. Syksyllä 2009 julkaistussa 2.6.28-versiossa kernelin ohjelmarivien määrä ylitti jo 10 miljoonaa[16]. Noin neljäsosa tästä koodista on vain tietyille prosessoriperheille tehtyä, loppu on geneeristä kaikissa arkkitehtuureissa ajettavaa ohjelmaa.



Kuva 10. Linuxin kernelarkkitehtuuri.

Kuvassa 10 on esitetty rakennemalli Linuxin kernelistä ja sen tärkeimmistä alijärjestelmistä. Kuvassa esitetty järjestelmäkutsurajapinta tarjoaa käyttäjän ohjelmille pääsyn kernelin palveluihin. Jokaisessa käytännön toteutuksessa on lisäksi kernelin fyysiseen laitteeseen liittävä laiteajuri-kerros. Laiteajurit on tehty helpottamaan ylemmän tason ohjelmien suunnittelua. Ne piilottavat elektroniikan ohjauksen helpommin käytettävän ja ymmärrettävän rajapinnan taakse. Laiteajuri koostuu kahdesta osasta, kaikille saman loogisen toiminnallisuuden tuottaville laitteille yhteisestä loogisesta osasta ja juuri kyseisessä laitteessa käytettävän elektroniikan ohjauksesta huolehtivasta fyysisestä osasta. Käytettävän elektroniikan muuttuessa joudutaan laiteajurin fyysistä osaa muuttamaan vastaamaan muuttunutta piirikytkentää.

Laiteajuri toimii siis sovituskerroksena, sovelluksen tai kernelin liittämiseksi fyysiseen laitteeseen. Laiteajurin looginen osa tulkitsee loogisille laitteille lähetetyt komennot, fyysinen kerros huolehtii varsinaisen elektroniikan ohjaamisesta toimimaan halutulla tavalla. Loogisia laitteita ovat esimerkiksi nopean sarjaliikenneyhteyden tarjoava USB- tai verkkoliitynnän tarjoava ETHERNET-liityntä.

Järjestelmäkutsurajapinta on joukko funktioita, jotka välittävät käyttäjän ohjelmien palvelukutsuja kernelille ja laiteajureille. Kutsussa vaihdetaan prosessorin tila käyttäjätilasta etuoikeutetuksi. Etuoikeutetussa kerneltilassa toimivat ohjelmat näkevät kaikki laitteen resurssit ja niillä on pääsy kaikkeen fyysiseen muistiin. Kontrollin palatessa käyttäjän ohjelmalle, tila vaihdetaan takaisin, jolloin käyttäjän ohjelma näkee vain oman rajatun muistialueensa.

Vaikka suurin osa Linuxista on käytettävästä prosessoriarkkitehtuurista riippumatonta, on jokaisella prosessoriperheelle myös sille spesifistä ohjelmakoodia. Tämä osio on kuvattu kaaviokuvassa yhdellä lohkolla. Tässä lohkossa on kyseisen arkkitehtuurin erityisvaatimukset toteuttavat ohjelman osiot. Ne saattavat liittyä esimerkiksi muistinhallintaan ja prosessorin alkukäynnistykseen.

Kernelin muiden osioiden tehtävät:

- Prosessien hallinta vastaa prosessien luonnista ja niiden vuorottamisesta. Prosessia luodessaan hallinta varaa prosessille resurssit. Linuxissa se huolehtii myös keskeytysten geneerisestä käsittelystä.
- Virtuaalinen tiedostojärjestelmä tarjoaa käyttäjälle yhteisen kutsurajapinnan kaikkiin tiedostojärjestelmiin. Näitä kutsuja ovat esimerkiksi ”open”, ”read”, tai ”write”-toiminnot. Runsaasti erilaisia tiedostojärjestelmiä[17], joista mobiililaitteissa käytetyimpiä ovat ext3[18] ja JFFS2[19]. Linux edellyttää yleisesti, että laitteistossa on muistinhallintayksikkö, joka tarjoaa laitetason suojauksen fyysiselle muistille. Linuxin muistinhallinta vastaa muistin dynaamisesta varauksesta ja se pitää kirjaa muistin käytöstä.
- Linuxin muistinhallinta huolehtii muistin dynaamisesta varauksesta ja vapauttamisesta.
- Verkkoliityntä implementoi IPv4- ja IPv6-protokollapinon[20][21].

2.6 Linux-projektin toiminta

Linuxin kernelin kaltaisen suuren ohjelmakokonaisuuden kehittäminen hallitusti

eteenpäin on haasteellinen tehtävä. Jotta kernelin kehitys olisi hallittavissa, on sen ylläpitovastuu jaettu toiminnallisiin alueisiin. Erikseen hallittavia kokonaisuuksia ovat esimerkiksi eri prosessoriarkkitehtuurit ja alijärjestelmät. Näitä alijärjestelmiä ovat fyysisten laitteiden ajurit ja abstraktit tiedostojärjestelmät. Kustakin tällaisesta kernelin osa-alueesta on vastuussa ylläpitäjä, jonka tehtävänä on hyväksyä tai hylätä kyseisen osion ohjelmakoodiin tulevat muutokset. Useimmissa tapauksissa kunkin alueen ensimmäinen ylläpitäjä on kyseisen ohjelmakoodin alun perin Linuxille kirjoittanut suunnittelija. Ylläpitäjä vaihtuu, kun joku aktiivinen kontribuuttori saa ylläpitäjän paikan sen edelliseltä haltijalta. Vaikka ylläpitäjä päättää lopullisesti, mitä ohjelmakoodia hänen alijärjestelmäänsä sisällytetään, on kuka tahansa oikeutettu kommentoimaan ehdotettuja muutoksia. Käytännössä ylläpitäjän tukena on joukko kokeneita ja luotettavia kommentoijia, jotka suorittavat ehdotettujen koodimuutosten katselmoinnin. Tämän katselmointituloksen perusteella ylläpitäjä tekee päätöksensä koodin hyväksymisestä. Ylläpitäjän tehtävänä on myös tarjota tehdyt muutokset käyttöjärjestelmän päähaaraan, Linus Torvaldsin ylläpitämään Linuxin julkiseen puuhun. Tätä päähaaraa kutsutaan sen ylläpitäjän mukaisesti "Linusin puuksi". Torvalds päättää mitä ominaisuuksia otetaan mukaan kuhunkin viralliseen versioon.

Halutessaan muuttaa kernelkoodia tai lisätä tähän uuden toiminnallisuuden, ohjelmoija kirjoittaa kernelin kehityksessä käytettävän GIT-versionhallinnan[23] alaisuuteen halutun muutoksen toteuttavan ohjelmakoodin. Versionhallinnan työkalulla tehdään erotiedosto kernelin edelliseen versioon nähden. Tämä tiedosto sisältää tiedon kaikista lisätyistä, poistetuista ja muutetuista riveistä. Erotiedosto lähetetään kyseisen alijärjestelmän postituslistalle kommentteja varten. Kyseisestä alueesta kiinnostuneet voivat rekisteröityä alueen sähköpostilistan tilaajiksi[24]. Kaikki tälle postituslistalle lähetetyt sähköpostit lähetetään edelleen listalle rekisteröityneille henkilöille. Näin kyseisestä kernelin alueesta kiinnostuneet voivat seurata kaikkea alueella tapahtuvaa kehitystä. Kuka tahansa on oikeutettu kommentoimaan listalle lähetettyä, kerneliin ehdotettua ohjelmakoodia. Kaikki kyseistä muutosta koskevat kommentit lähetetään sähköpostilistalle vastauksena alkuperäiseen muutosehdotukseen. Alijärjestelmän tuntevat kokeneet Linux-suunnittelijat kommentoivat yleensä oman alueensa muutoksia. Kun muutoksen tekijä on korjannut ohjelmakoodiaan esitysten mukaan, hyväksytään muutos osaksi alijärjestelmää ja ylläpitäjä liittää sen osaksi alijärjestelmäänsä. Kyseisestä alijärjestelmästä muutokset siirtyvät vaiheittain Linuxin pääpuuhun. Tämän jälkeen jokainen uuden kernelin lataaja saa tehdyn muutoksen osana vakaata kernelkoodia. Tullakseen hyväksytyksi, on muutoksen täytettävä kernelkoodille

asetetut vaatimukset. Nämä vaatimukset koskevat koodin toiminnallisuutta, ulkonäköä ja arkkitehtuuria. Tarkat vaatimukset löytyvät Linuxin kernelissä mukana tulevasta tekstitiedostosta[25].

Kernelin kehitysprosessia kutsutaan Linux-yhteisössä koodin kontribuoinniksi. Kaikki postituslistat ja niihin liittyvä ohjeistus löytyvät kernel.org-sivustolta. Linux-kernel julkaistaan kokonaisuudessaan GPLv2-lisenssillä[14]. Mobiililaitteiden ohjelmiston kehittämisen kannalta tämä tarkoittaa sitä, että ohjelmoijan tuottama kernelkoodi on julkaistava kaikkien halukkaiden saataville. Tämä voi tapahtua kolmella eri tavalla.

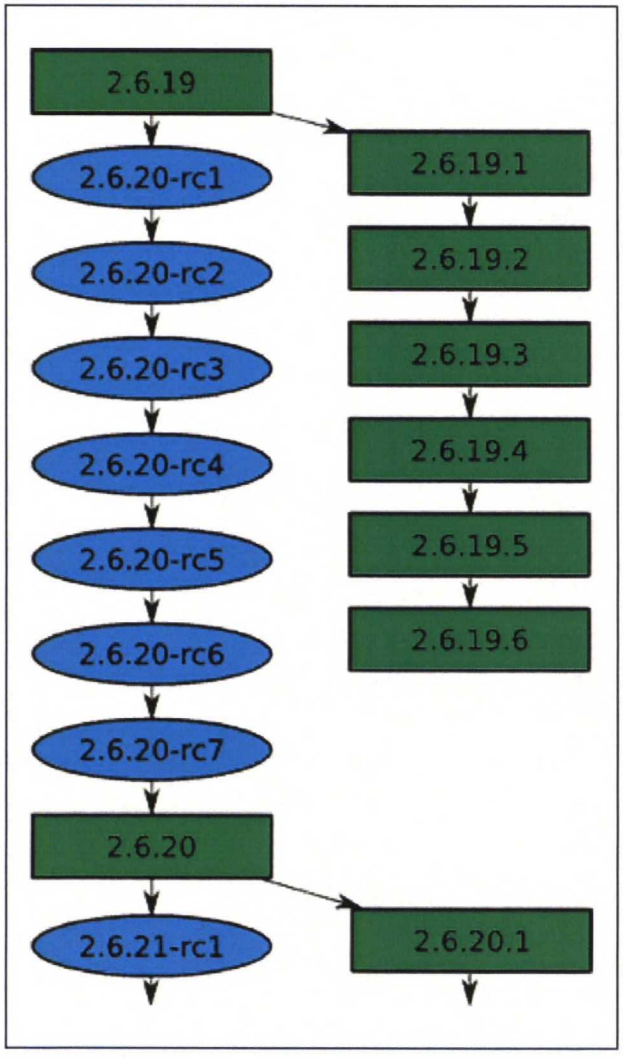
- Tuotettu kernelkoodi voidaan projektin jälkeen asettaa vapaasti halukkaiden saataville. Tämä onnistuu esimerkiksi asettamalla se yrityksen avoimille verkkosivuille.
- Tuotettu koodi voidaan projektin valmistumisen jälkeen liittää osaksi julkista Linux-kerneliä. Tämä tapahtuu noudattamalla aiemmassa kappaleessa kuvattua kontribuointiprosessia.
- Projekti voi tehdä kernelkehityksensä kiinteässä yhteistyössä Linux-yhteisön kanssa. Tällöin valmiit kernelkoodin osat kontribuoidaan Linuxin kerneliin heti valmistumisen jälkeen.

Viimeisin tapa kolmesta vaihtoehdosta on vaativin, mutta samalla se tuo Linux-yhteisön hyödyn parhaiten projektin käyttöön. Kiinteä yhteistyö vaatii myös yleensä yrityksen sisäisen tuotekehityksen toimintamallin muuttamisen. Joustava yhteistyö vaatii yritystä käytännössä käyttämään sisäisessäkin kehityksessä Linux-kernelin kehitystapaa ja työkaluketjua. Tähän työkaluketjuun kuuluu GIT-versionhallinta ja GNU-kääntäjien käyttäminen.

Linuxin kernelin versionumero on tämän työn kirjoittamisen aikaan 2.6.33. Kernelversioden numerointi koostuu kolmesta osiosta.

- Pääversionumero, varsinainen versio, joka tällä hetkellä on 2. Nykyinen versio on julkaistu vuonna 1996.
- ”Major revisio”-numero, joka tällä hetkellä on 6. Versio 2.6 on julkaistu vuonna 2004.
- ”Minor revisio”-numero, joka vaihtuu noin kahdentoista viikon välein, tällä hetkellä viimeinen versio, 2.6.33, on julkaistu 26.4.2010[22].
- Näiden varsinaisten kernelversion numeroiden lisäksi käytetään mahdollisten

korjausten tuottamia versionumeroita, esimerkiksi 2.6.33.1, ja julkaisukandidaattinumeroita kuten 2.6.34.rc1



Kuva 11. Linuxin kernelversion kehitys.

Kuvassa 11 on esitetty kernelversion muutos versiosta 2.6.19 versioon 2.6.20. Kun kernelin edellinen versio on julkaistu, kuvassa 2.6.19, kootaan seuraavan parin viikon aikana kaikki ne muutokset, jotka on hyväksytty kernelkoodiin. Nämä integroidaan uusimpaan viralliseen versioon, 2.6.19. Tätä kutsutaan julkaisukandidaatiksi, joka tunnistetaan .rc1-tarkenteesta. Näin version 2.6.19 jälkeen seuraava epävirallinen versio, julkaisukandidaatti, on 2.6.20-rc1. Tämä tarkoittaa version 2.6.20 ensimmäistä kandidaattia. Tätä julkaisukandidaattia testataan ja siihen tehdään testauksen perusteella muutoksia kernelin stabiloimiseksi. Näitä .rc-versioita kootaan, kunnes on

aika julkaista seuraava vakaa kernelversio. Tähän uuteen versioon otetaan mukaan vain ne osiot, jotka toimivat moitteetta, ja tälle versiolle annetaan esimerkkitapauksessa nimeksi 2.6.20. Ne ohjelmakoodin osat, joita ei ole saatu maturoitua, siirretään seuraavaan julkaisukandidaattiin. Tästä alkaa jälleen seuraavan kernelversion kehitys. Seuraava kandidaattiversio on 2.6.21-rc1.

Kandidaattijulkaisujen lisäksi joudutaan varsinaiseen stabiiliin versioon usein tekemään korjauksia. Ensimmäinen tällainen korjattu versio olisi edellä kuvatussa tapauksessa 2.6.19.1. Stabiilia versioa maturoidaan, kunnes seuraava stabiili versio on julkaistu.

2.7 Avoimen lähdekoodin lisenssit

Tietokoneohjelman tekijällä on lakisääteinen oikeus työhönsä. Tämä oikeus suojaa teoksen toteutustavan mutta ei itse toteutuksen ideaa. Tämän ehdon vuoksi kuka tahansa voi toteuttaa vastaavan toiminnallisuuden tuottavan ohjelman ja saada siihen tekijänoikeudet. Ohjelman tekijä voi julkaista kirjoittamansa ohjelmakoodin haluamallaan ohjelmistolisenssillä varustettuna. Ohjelmistolisenssi tarkoittaa niitä ehtoja, joilla lisenssin alaista ohjelmistoa saa käyttää, muuttaa, liittää muihin ohjelmistoihin tai levittää eteenpäin.

Avoimen ohjelmakoodin yhteydessä käytetään useita erilaisia ohjelmistolisenssejä. Tunnetuimpia näistä ovat GPL[14], LGPL[26], MIT[27], FreeBSD[28] ja Apache[29]-lisenssit. Kaikille avoimen koodin lisensseille on yhteistä niiden antama lupa käyttää, tutkia ja jakaa eteenpäin lisenssin alaista ohjelmakoodia, joko alkuperäisessä muodossa tai muutoksia sisältävänä. Näistä oikeuksista ei avoimen ohjelmistolisenssin yhteydessä joudu maksamaan lisenssimaksuja. Mikäli ohjelmakoodia käyttää vain yksityisesti, eikä levitä sitä eteenpäin, ei muutettua koodia tarvitse myöskään julkaista. Lisenssit erottaa toisistaan se, miten muutettuja versioita saa edelleen levittää. Niin sanotut "copyleft"-lisenssit vaativat, että alkuperäinen lisenssi säilytetään kaikissa lisensioituun ohjelmistoon perustuvissa, edelleen levitettävissä ohjelmissa. Jos copyleft-lisensioitua ohjelmakoodia käyttää osana laajempaa ohjelmistoa, on koko ohjelmisto julkaistava alkuperäisellä lisenssillä varustettuna.

Avoimen koodin lisensseistä GPL-lisenssit vaativat tiukimmin koodin pysymistä avoimena. Ne ovat copyleft-lisenssejä, jotka edellyttävät kaiken niistä periytyvän ohjelmakoodin julkaisemista saman GPL-lienssin alaisuudessa. Mikäli ohjelmakoodia käytetään kaupallisiin tarkoituksiin, on ohjelman lähdekoodi ja sen käyttölupa annettava niitä haluaville tahoille ilman erillistä korvausta. GPLv2 vaatii jopa siihen linkitettävien

ohjelmistojen muuttamista GPLv2:n mukaiseksi. Lesser GPL, LGPL-lisenssi, sallii dynaamisesti linkitetyn koodin käytön toiminnallisuuden laajentamiseksi siten, että linkitetty ohjelmakoodi voi olla jollakin muulla lisenssillä julkaistua. Staattisessa linkityksessä kaikki moduulit, joihin ohjelma viittaa, liitetään ohjelman suoritusaikaiseen versioon. Dynaamisesti linkitettävät ohjelmat ovat omana ajettavana tiedostonaan, ja ne ladataan suoritettavaksi vasta tarvittaessa. LGPL-lisensoidun koodin voi muuttaa GPL-lisensoiduksi mutta ei toisinpäin.

Linux käyttää kernelissä Richard Stallmanin alun perin vuonna 1989 luomaa GPL-lisenssiä[14]. Kernelin ensimmäisten versioiden valmistumisen aikaan lisenssin uusin versio, GPLv2, otettiin käyttöön Linuxissa. Tämä lisenssi on säilynyt Linux-kernelin käytössä. Linuxin kernelin yhteydessä on erityisesti huomioitava, että käytettäessä lisensioitua koodia kaupallisiin tarkoituksiin, edellytetään myös, että ohjelman lähdekoodi annetaan kaikkien halukkaiden saataville.

Apache ja FreeBSD ovat tunnetuimpia niistä avoimista lisensseistä, jotka eivät vaadi ohjelmakoodin julkaisua tai alkuperäisen lisenssin säilyttämistä levitetyn ohjelmakoodin mukana. FreeBSD on myös avoimen koodin perustuva täydellisen käyttöjärjestelmän tarjoava ohjelmisto. Se pohjautuu Berkeleyn yliopiston versioon Unix-käyttöjärjestelmästä[30]. Ohjelmiston kernelosa on julkaistu FreeBSD-lisenssin alla. FreeBSD:n lisäksi käytettävissä on muitakin BSD-UNIX:iin pohjautuvia avoimen ohjelmakoodin lisenssejä. FreeBSD:lle kirjoitetut käyttäjän ohjelmat, kuten kaikkien Unix-pohjaisten käyttöjärjestelmien ohjelmat, toimivat yleensä ilman muutoksia myös Linuxissa. FreeBSD- lisenssi sallii ohjelman vapaan kopioimisen ja käytön. Se rajoittaa uudelleenjulkaisua ja muutetun koodin käyttöä vähemmän kuin Linuxin käyttämä GPL-lisenssi, sallimalla linkittämisen myös suljetun lisenssin ohjelmien kanssa. FreeBSD-lisenssi sallii myös ohjelman levittämisen binäärikoodina, koodin on tässä tapauksessa kuitenkin sisällettävä tarkkaan määrätty lisenssistä kertova lauseke. Alkuperäistä tai muutettua lähdekoodia ei ole välttämätöntä julkaista FreeBSD:n tapauksessa. Apache-lisenssi on alun perin kirjoitettu Apache-web-serveriä varten. Lisenssin uusin versio, 2.0, on erittäin suosittu avoimen koodin ohjelmaprojektien lisenssinä. Se sallii että ohjelmakoodista periytyvät ohjelmat julkaistaan eri lisenssillä. Apache-lisenssille tunnusomaista on sen vaatimus säilyttää kaikki alkuperäiset tuotemerkki-, patentti- ja tekijänoikeusmerkinnät kaikista ohjelmistossa olevista koodeista.

2.8 Avoimen lähdekoodin mukaanoton haasteet tuotekehityksessä

Linux-käyttöjärjestelmän käyttöönotto asettaa tuotekehitysorganisaation uusien haasteiden eteen. Monet vanhat ja totutut menettelytavat on korvattava uusilla. Tuoteprojektin johdon täytyy ymmärtää GPLv2-lisenssin vaatimukset. Nämä vaatimukset saattavat rajoittaa yrityksen oman ohjelmistosalkun käyttöä. Kaikesta kernelissä käytettävästä ohjelmakoodista tulee julkista ja kuka tahansa saa GPL-lisenssin mukaisesti kopioida, muuttaa ja käyttää kyseistä ohjelmakoodia. Tällöin kernelissä käytetty koodi avataan kenen tahansa käyttöön ja siihen kohdistuva immateriaalioikeus menettää käytännössä merkityksensä. Koodin julkaiseminen tuotekehityksen aikana saattaa myös paljastaa kilpailijoille tekeillä olevan laitteen ominaisuuksia.

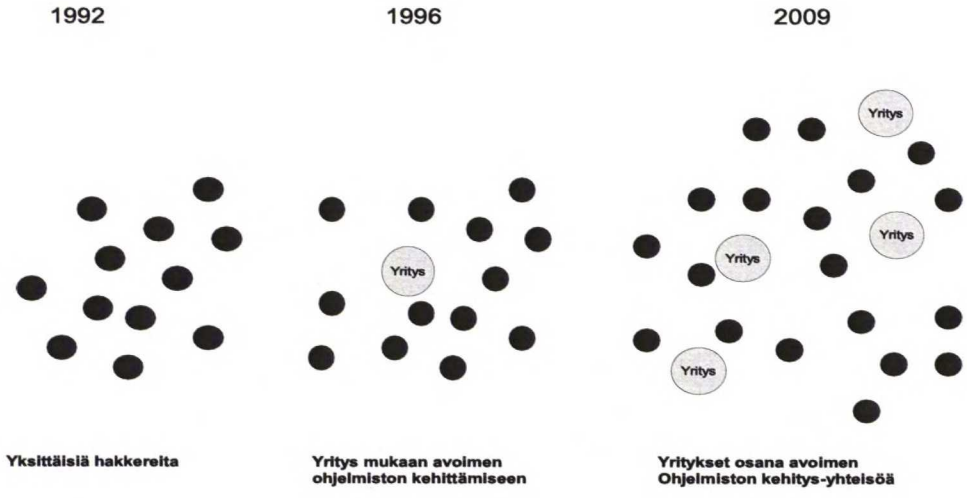
Avoimeen ohjelmakoodiin perustuvan käyttöjärjestelmän käyttöönotto asettaa yleensä suurimmat haasteet itse ohjelmointityötä tekevien suunnittelijoiden osaamiselle. Heidän on opittava kokonaan uudenlaiset työmenetelmät ja ohjelmointiympäristö. Linux-kernelin tapauksessa tämä tarkoittaa kernelin liityntöjen, versionhallinnan ja tuotettavan ohjelmiston käännösprosessin oppimista. Mikäli Linuxin käytöstä halutaan saada yritykselle mahdollisimman suuri hyöty, on näiden varsinaiseen ohjelmointityöhön vaikuttavien muutosten lisäksi opittava yhteistyö Linux-yhteisön kanssa. Kunkin ohjelmoijan on tutustuttava kernelin ja erityisesti oman osaamisalueensa kehitykseen. Linux-kernelin kehitystyö on jatkuvaa ja hajautettua. Kaikki kerneliä käyttävät tahot voivat osallistua yhteisön toimintaan, tuomalla oman osaamisensa ja työpanoksensa mukaan. Ohjelmoijan osaamisen muuttuneet vaatimukset koskevat niin yrityksen omia työntekijöitä, kuin projektissa käytettäviä alihankkijoitakin. Kokenut ohjelmoija oppii Linuxin kernelin pääpiirteet ja sen ohjelmointiympäristön muutaman kuukauden harjoittelun jälkeen.

2.9 Aiemmat tutkimukset

Tämän diplomityön aiheesta, yhden projektin saamasta hyödystä sen toimiessa kiinteässä yhteistyössä Linux-yhteisön kanssa, ei ole aiemmin tehty julkista tutkimusta. Useat tutkimukset lähestyvät tätä aihetta eri suunnista, mutta ne keskittyvät pidempiaikaisiin yrityksen ja ohjelmoijan saamiin hyötyihin. Tämän tutkimuksen näkökanta eroaa näistä aikajänteen osalta. Yhden projektin kannalta pitkällä tähtäimellä saavutetuilla eduilla ei ole merkitystä.

Linuxiin liittyviä tutkimuksia on tehty yli kymmenen vuotta. Merkittävä osa niistä

keskittyy avoimen ohjelmakoodin käyttöön yleisemmin ja tarkastelee käytön merkitystä liiketaloudelliselta kannalta. Pohdinnat koskevat paljolti avoimen koodin käyttöä osana strategiaa ja toisaalta käytön tuomia haasteita. Haasteet liittyvät paljolti avoimen koodin GPL-lisenssin tuomiin haasteisiin. Toinen osa tutkimuksista kohdistuu avoimen koodin projekteihin osallistuvien ohjelmoijien motiiveihin. Varsinkin Linuxin kehityksen alkuaikoina suuri osa kehittäjistä teki työtä ilman suoraa taloudellista hyötyä. Näiden kahden tutkimushaaran väliin jää alue, jossa tutkitaan yrityksen ja Linux-yhteisön yhteistyön merkitystä yritykselle. Kuvassa 12 on esitetty Linux-yhteisön rakenteen muutos.



Kuva 12. Linuxin kehittäjäyhteisön muutos hakkerikeskeisestä yritykset käsittäväksi.

Avoimen koodin ja Linuxin käytön yritykselle tuomia hyötyjä on tutkittu paljon. Ne eivät niinkään käsittele yksittäisen projektin saavuttamia lyhytaikaisempia etuja, vaan keskittyvät enemmän pitkäjänteisen yhteistyön tuomiin hyötyihin. Tutkimuksessaan vuodelta 2004 Dahlander ja Magnusson tulivat johtopäätökseen, jonka mukaan aktiivisesti avoimien ohjelmistojen kehitykseen osallistuvat yritykset pystyivät vaikuttamaan yhteisön projekteihin[31]. Mikäli yritys kykenee vaikuttamaan avoimienohjelmien kehitykseen, se voi ohjata avoimen koodin projekteja tukemaan

yrityksen omia tavoitteita. Uraauurtavassa kirjassaan "Cathedral and Bazaar" näkee Eric Raymond avoimen ohjelmakoodin tuottamisen strategisesti tärkeänä aseena kaupalliselle yritykselle[32]. Kirjassa todetaan avoimeen ohjelmakoodiin perustuvan Apache-verkkopalvelimen kehittämisen olleen strateginen päätös estää Microsoftia saamasta monopolia omalla verkkopalvelintoteutuksellaan. Useiden yritysten yhteinen projekti tuotti maksuttoman avoimeen koodiin perustuvan vaihtoehdon, joka nopeasti syrjäytti Microsoftin tarjoaman ratkaisun. Täten avoimen koodiin perustuvien projekteihin osallistuminen auttaa omalta osaltaan estämään monopolien syntyä. Kun jokin ohjelma saa hallitsevan aseman markkinoilla, on yleisesti ottaen toivottavaa, että tämä ohjelma ei ole yhdenkään yrityksen omaisuutta. Lisäksi yksittäisen ohjelmoijan Linux-yhteisön yhteistyöstä saama hyöty palvelee myös yrityksen tavoitteita. Henkilöstön kompetenssin kasvu ja motivaatio ovat tärkeitä tekijöitä tuottavuudelle. Linux-yhteistyöstä syntyneiden suhteiden avulla on yrityksen helppo löytää osaavaa henkilökuntaa.

Yksittäiselle ohjelmoijalle avoimen koodin käyttö tuo monia etuja. Toisin kuin suljettua ohjelmaa kirjoitettaessa, saa avoimen ohjelmakoodin kehittäjä nimensä julkisuuteen kontribuoidessaan koodin. Tämä voi monelle olla tärkeää omanarvontunnon kannalta ja toimii julkisena osoituksena ammattitaidosta[33]. Eniten kontribuutioita tehneiden nimet ovat kaikkien kehittäjien ja rekrytoijien nähtävillä vuotuisissa Linux-konferensseissa. Työpaikkaa hakiessa voi myös helposti osoittaa osaamisensa, viittaamalla tekemiinsä ohjelmakoodeihin Linuxin kernelissä. Tämän tutkimuksen kirjoitushetkellä Linuxin kernelin kehitys on siirtymässä enenevissä määrin yritysten harteille, nykyisin yli 75 % kaikesta uudesta kernelkoodista kehitetään yrityksien maksamissa projekteissa [34]. Tämä on muokannut Linux-kehitysyhteisön enemmän ja enemmän yrityspainotteiseksi. Linuxiin liittyvästä osaamisesta on tullut kysytty ominaisuus ja Linuxin kernelin kehitystyötä tekevät yritykset käyttävät tietoja kontribuutioita tehneistä kehittäjistä etsiessään työntekijöitä.

Tehtäessä kyselyjä avoimen koodin kehittäjien motiiveista[35], havaittiin tärkeimmiksi syiksi oman kompetenssin kasvu ja mielenkiinto työtä kohtaan. Linux-yhteisöön verkottumisen ja kernelin laadun parantamisen katsottiin myös olevan tärkeitä motivaattoreita. Henckel ja Tins[36] totesivat tutkimuksessaan tärkeiksi motiiveiksi myös oman uran mahdollisuuksien paranemisen ja Linuxia käyttävien kuluttajatuotteiden hinnan laskemisen. Merkittävässä tutkimuksessaan Lerner ja Tirole[37] toteavat odotukset tulevista taloudellisista eduista yhdeksi pääsyyksi ohjelmoijien kiinnostukselle avoimen koodiin pohjautuviin projekteihin. Osallistumalla

projekteihin kehittäjät osoittavat kykynsä mahdollisille työnantajille.

Feller, Hissam ja Lakhani jakavat ohjelmoijien motivaation neljään luokkaan[38].

1. Ne ohjelmoijat, joiden motiivina on uuden tekeminen ja uusien asioiden oppiminen.
2. Henkilökohtaista etua hakevat, jotka katsoivat Linux-kontribuutioissa saadun julkisuuden tarjoavan mahdollisuuksia edetä uralla.
3. Linuxin koodia omiin tarpeisiin tekevät. Tämä ei ole projektissa työskentelevien motiivina kuin erityistapauksissa.
4. Ohjelmoijat, jotka kannattavat avoimen ohjelmiston käyttöä ideologisista syistä.

Yleisesti ottaen kaikissa tutkimuksissa ohjelmoijan Linux-kontribuoinnille löytämät motiivit olivat samansuuntaisia. Projektin ulkopuolisten henkilöiden motiivit tehdä työtä ilman suoraa korvausta ovat hyvin samanlaisia.

3. Käytetty tutkimusmenetelmä

Tässä kappaleessa esitetään ensin tutkittavan tiedon tuottaneet työkalut, työn lähdemateriaalina oleva tieto ja sen rakenteet. Sen jälkeen käydään läpi analysoinnissa käytetyt ohjelmistot ja esitellään asiat, joiden toteuttaminen on vaatinut manuaalisia vaiheita.

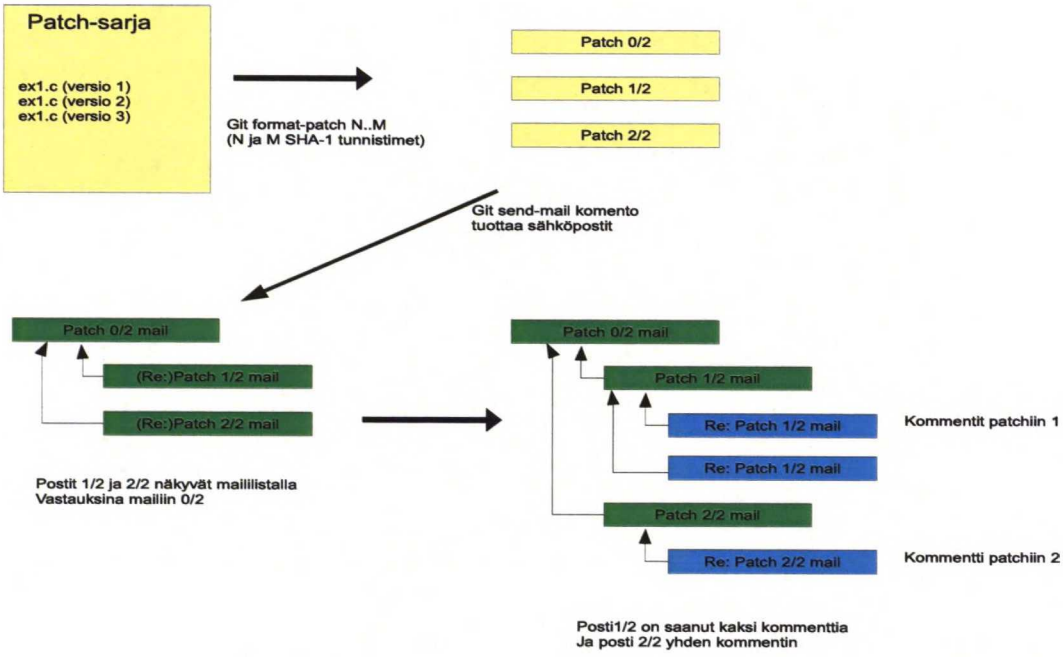
3.1 GIT-ohjelman toiminta

Linuxin kernelin kehityksessä käytetään tavallisesti GIT-versionhallintaohjelmistoa[23], joka on suunniteltu erityisesti hajautettua ohjelmankehitystä varten. GIT:in käyttö perustuu tunnettuun lähtökohtaan, jonka kehittäjä kopioi omalle koneelleen ja jota vasten muutokset tehdään. Kukin ohjelmoija testaa omat muutoksensa tätä lähtökohtaa vastaan. Tehdessään muutoksia Linuxin kerneliin, ohjelmoija ensin suunnittelee ja testaa uuden ohjelmakoodin. Kun tehdyt muutokset toimivat ohjelmoijan vaatimusten mukaisella tavalla, lähettää hän sen soveltuvalle sähköpostilistalle Linux-yhteisön kommentteja varten. Ehdotettua muutosta kutsutaan nimellä "patch". Linux-yhteisön asiantuntijat, käytännössä kyseisen ohjelma-alueen muut suunnittelijat, kommentoivat tehtyjä muutoksia. Mikäli uudessa ohjelmakoodissa on jotain parannettavaa, korjaa ohjelmakoodin suunnittelija ohjelmakoodin vastaamaan yhteisön vaatimuksia. Kun ohjelmakoodi on hyväksytty, liitetään se kyseiseen alijärjestelmään ja julkaistaan seuraavassa kernelversiossa.

Kernelkontribuutioissa edellytetään, että ehdotettu muutos, patch, lähetetään postituslistalle muodossa, joka on yhteensopiva GIT-versionhallinnan[23] tuottaman tiedoston kanssa. GIT-komennoilla aikaansaatu erotiedosto kuvaa yksiselitteisesti erot kahden eri version välillä. Uusi versio tiedostosta viedään versionhallintaan "git commit"-komennolla. Näin muodostettu erotiedosto, patch, kuvaa yksiselitteisesti mitä muutoksia kyseinen tiedosto Linux-kerneliin aiheuttaa. Tämän lisäksi patch sisältää 7-merkkisen osan, sekä alkuperäisen että muutetun tiedoston, SHA-1 tarkisteesta. Näin varmistetaan tiedon eheys, kun patchiä integroidaan Linuxin kerneliin.

Tavallisesti koodimuutokset ovat yksinkertaisia, yhteen tiedostoon tai alueeseen kohdistuvia. Joissakin tapauksissa tehdyt muutokset kuitenkin muodostavat laajan kokonaisuuden, jolloin ne on lähetetty patch-sarjana. Kuvassa 13 on esitetty tällainen tapaus. Keltaisella on kuvattu alkuperäisiä erotiedostoja, joista tehdään vihreinä viesteinä kuvattu patch-sarja. Sarjalla on yksi yhteinen sähköposti, joka esimerkissä on

”patch 0/2 mail”. Tässä sähköpostissa ohjelmoija kuvaa tekemänsä muutokset. Varsinaiset ohjelmakoodin muutokset ovat sähköposteissa ”patch 1/2 mail” ja ”patch 2/2 mail”. Patcheihin kohdistuneet kommenttiviestit on kuvassa esitetty sinisellä värillä. Ne kohdistetaan yleensä erikseen kuhunkin muutokseen. Analysoinnin kannalta on tällaisessa tapauksessa helpointa ryhmittää kaikki muutokset omiin säikeisiinsä, jotta ohjelmakoodiin kohdistuneet kommentit laskettaisiin oikein. Kommenttiviestit hajautuvat eri muutoskohtia koskeviksi säikeiksi. Mikäli ”patch 0/2 mail” saa kommentteja, lasketaan ne kohdistuneeksi kaikkiin sarjan koodimuutoksiin.



Kuva 13. Usean patchin tuottaminen ja kommentointi postituslistalla

Seuraavassa esitetään tyypillinen työsekvenssi muutoksen tekemiseksi GIT-versionhallinnassa olevaan tiedostoon ja kontribuutio-sähköpostin tekemiseksi muutoksesta. Alkuperäinen c-kielinen tiedosto, johon muutokset tehdään, on versionhallinnassa nimellä ex1.c. Tämä tiedosto sisältää yksinkertaisen ohjelman, jonka pituus on 8 riviä. Ohjelmatiedosto avataan emacs-ohjelmalla editoitavaksi. Kyseisestä ohjelmatiedostosta poistetaan yksi tyhjä rivi ja yhtä riviä muutetaan (”Testing” → ”Testing again”). Tämän jälkeen muutoksesta generoidaan erotiedosto. Näin syntynyt erotiedosto lähetetään sähköpostin runkoon kopioituna halutulle postituslistalle.

Alkuperäinen tiedosto, johon muutokset tehdään:

```
#include <stdio.h>
main()
  {
    printf ("Hello world\n");
    printf ("Testing\n");


  }
```

Otetaan alkuperäinen tiedosto versionhallinnasta

```
kosh Dt 19 % git checkout ex1.c
```

Käynnistetään editori ja avataan ex1.c-tiedosto

```
kosh Dt 20 % emacs ex1.c
```

Muutetaan jälkimmäistä printf-lausetta, ja poistetaan sen jälkeinen tyhjä rivi. Talletetaan tehdyt muutokset.

Tiedosto tehtyjen muutosten jälkeen:

```
#include <stdio.h>
main()
 {
   printf ("Hello world\n");
   printf ("Testing again\n");
 }
```

Talletetaan muutos GIT-versionhallintaan.

```
kosh Dt 22 % git commit -a
```

Tehdään erotiedosto kontribuointia varten. Komento tekee erotiedoston kahden viimeisen version välille.

```
kosh Dt 23 % git format-patch HEAD^..HEAD
```

Edellisellä komennolla tuotettu tiedosto, joka on valmis lähetettäväksi sähköpostilla:

```
From be754c91708221739b7d0d8d7aa68511fa83b9e1 Mon Sep
17 00:00:00 2001
From: =?utf-8?q?Altti=20Hell=C3=A4koski?=
<alttih@cc.hut.fi>
Date: Thu, 18 Feb 2010 13:06:06 +0200
Subject:[*PATCH] Altered version of ex1.c

---
 ex1.c |    3 +--
 1 files changed, 1 insertions(+), 2 deletions(-)

diff --git a/ex1.c b/ex1.c
index e898d22..b997995 100644
--- a/ex1.c
+++ b/ex1.c
@@ -3,6 +3,5 @@
 main()
 {
     printf ("Hello world\n");
-    printf ("Testing\n");
-
+  printf ("Testing again\n");
    }

--

1.6.0.4
```

Huomioi, että erotiedostossa mukaan tulee myös kolme muutoskohtaa edeltävää ja sitä seuraavaa koodiriviä oikean kohdan varmistamiseksi.

Käytännön tapauksessa toiminnallinen muutos vaatii usein muutoksia useaan kohtaan ohjelmakoodia. Tällaiset muutokset lähetetään yhtäaikaisesti patch-sarjana. GIT-työkalulla voi tehdä tällaisen sarjan. Komennossa annetaan tällöin kaikkien niiden versioiden SHA-1-tunniste, joiden välillä erotiedostot tehdään. Tällöin jokaisesta väliversiosta syntyy oma erotiedosto. Tämä sallii jokaisen muutoksen erillisen kommentoinnin. Erotiedostojen lisäksi syntyy ”patch 0/N” (N erotiedostojen määrä), joka on kaikille joukon erotiedostoille yhteinen kommentti-tiedosto, jonne tekijä kirjoittaa muutoksien syyn ja sisällön. Jokaisesta muutoksesta tulee sähköpostilistalle oma kommentointisäie. Menettely on esitetty kuvassa 13.

Edellä kuvattu erotiedosto lähetetään katselmointia varten, kommenteilla lisättynä, soveltuvalle postituslistalle. Sähköpostin tulee noudattaa RFC2822:n määrittelemää

rakennetta[39]. Kyseisen spesifikaation määrittelemiä kenttiä käytettiin tässä työssä identifioimaan tutkittava sähköposti. Mikäli sähköposti ei noudattanut RFC 2822:a, se tilastoitiin hylättyihin sähköposteihin.

3.2 Analysoitava tieto

Ensimmäiset Nokia N900-laitetta koskevat kontribuutiot etsittiin linux-omap-postituslistalta käsin. Ne oli lähetetty sinne maaliskuun 25. päivänä 2008. Tämä päivämäärä valittiin analyysien tarkasteluajan alkupisteeksi. Laite julkistettiin USA:ssa 18.11.2009, joka valittiin tarkastelujakso loppuajankohdaksi. Tarkastelujakso käsittää siis 603 päivää. Kaikki tuolla välillä omap-linux-postilistalla ollut keskusteluliikenne analysoitiin. Kyseiseltä ajanjaksolta linux-omap-sähköpostitietokannasta löytyi yhteensä 19598 yksittäistä viestiä.

Analyyseissa on tarkasteltu sähköpostiviestejä kolmella postituslistalla. Havaintojen mukaan osa patcheistä lähetetään useammalle postituslistalle, mahdollisimman suuren katselmointikattavuuden saavuttamiseksi. Tämä tarkoittaa sitä, että koodimuutosehdotus lähetettiin näissä tapauksissa spesifisen, liityntäaluekohtaisen postituslistan lisäksi, myös geneeriselle linux-omap-listalle. Vaikka asiantuntijat todennäköisesti tutustuvat kyseiseen koodimuutokseen molemmilla postituslistoilla, lähettävät he kommentit yleensä vain toiselle niistä. Tällaisissa tapauksissa on huomioitava se mahdollisuus, että kahdelle postituslistalle lähetetty koodimuutos vääristää tilastointia. Tämän vuoksi eri postituslistoilla olevat, samaan patchiin kohdistuneet kommentit on käsitelty niin, kuin ne kaikki olisivat kohdistuneet vain linux-omap-listalla olevaan patchiin. Käytännössä tämä on tehty yhdistämällä kaikki sähköpostilistat yhdeksi tarkasteltavaksi kokonaisuudeksi, jossa duplikaatit on helppo huomata, ja vastaukset yksinkertaista koota samaan säikeeseen. Muutosehdotukset identifioitiin vain linux-omap-listalta ja muilta listoilta etsittiin vain näiden viestien duplikaatteihin tulleita kommentteja. Duplikaattien tunnistamisessa käytetään patchin luomisen yhteydessä generoitua SHA-1-tarkistetta. Koodimuutosehdotukset, joihin ei ole tullut yhtään kommenttia, tilastoitiin omaksi ryhmäkseen.

Koodin kehittämisessä käytetyt sähköpostilistat on arkistoitu gmane.org-verkko-palveluun[24]. Tarvittavat sähköpostit ladattiin arkistosta analyysien suorittamiseksi työkoneelle.

Työssä tutkittiin seuraavia kaupallisen projektin saamia hyötyjä:

- Julkaistessaan kernelohjelmiston muutokset virallisten postituslistojen kautta,

saa projekti osaavien Linux-ohjelmoijien katselmoinnin tekemälleen työlle. Tämän merkitys voidaan arvioida suoraan katselmoinnin kohteena olleiden ohjelmakokonaisuuksien rivimäärien ja saatujen kommenttien perusteella.

- Projekti saa avointa Linuxin kernelin ohjelmakoodia kehittävältä yhteisöltä valmista ohjelmakoodia. Tämän määrä voidaan laskea suoraan tutkimalla kontribuutiosähköposteja.
- Avoimen keskustelukanavan muut merkitykset. Kontribuutioissa käytetyt sähköpostilistat toimivat keskustelukanavina yhteisön asiantuntijoiden ja projektiin osallistuvien suunnittelijoiden välillä. Tätä keskustelukanavaa voidaan hyödyntää kahdensuuntaisesti, kysymään asiantuntijoiden neuvoja Linuxiin liittyvistä asioista ja antamaan ennakkoon erityistietoja projektin valmistamasta laitteesta. Näillä ennakkotiedoilla voidaan varmistaa, että tulevat kenrel-muutokset ovat mahdollisimman hyvin yhteensopivia tehtävän tuotteen kanssa. Hyöty voidaan arvioida laskemalla näiden keskustelujen lukumäärä.

Tässä työssä analysoitiin niitä Linux-kernelin muutosehdotuksia, jotka lähetettiin joko linux-omap-, linux-usb- tai linux-mmc-postituslistalle. Analyysissa käsiteltiin ainoastaan Nokia N900-tuotteen kernelkoodiin kohdistuneita muutoksia. Työssä tutkittiin ainoastaan julkisesti saatavia tietoja, eikä muuta mahdollista yhteistyötä projektin ja Linux-yhteisön välillä voitu analysoida. Sähköposteja analysoitaessa on oletettu, että annettuna aikavälinä suurin osa postituksista on käsitellyt joko OMAP2:n tai OMAP3:n lohkoja, jotka ovat lähes samoja molemmissa piireissä[10]. Täten OMAP2:sta koskevat parannukset ovat olleet myös OMAP3:lle relevantteja parannuksia. Listalle on lähetetty myös OMAP1- ja OMAP4-aiheisia viestejä, jotka eivät todennäköisesti liity OMAP3-kehitystyöhön. Viestien aihekenttien perusteella näitä OMAP1- tai OMAP4 -aiheisia viestejä on kuitenkin vain 3,6% kaikista viesteistä. Näiden viestien vähäisen lukumäärän vuoksi todettiin, että enimmäkseen listan liikenne tutkitulla aikavälillä, on ollut tutkitun projektin kannalta relevanttia liikennettä.

Muilta tutkittavilta listoilta, linux-mmc ja linux-usb, käsiteltiin ainoastaan ne viestit, jotka oli lähetetty myös linux-omap-listalle. Muille kuin linux-omap-listalle lähetettyjen viestien määrä on suhteellisen pieni, vain muutama sata viestiä, verrattuna linux-omap-listan noin 20000 viestiin. Näin ollen muiden kuin linux-omap-listan täydellisellä analysoinnilla ei saavuteta mainittavaa lisäinformaatiota. Muut listat otettiin analysointiin mukaan, koska ne sisältävät duplikaatteja linux-omap-listalle lähetetyistä kontribuutioista, ja ovat siten saaneet kommentteja toisia listoja seuraavilta

asiantuntijoilta. Näin menetellen suunnittelija saa koodistaan palautetta, sekä erityisalueeseen perehtyneiltä asiantuntijoilta että OMAP3-toteutukseen osallistuvilta.

Jaottelukriteerinä käytettiin sähköposteissa olevia verkkotunnisteita. Varsinainen projektihenkilöstö lähetti viestit nokia.com-sähköpostipalvelimelta. Kiinteästi projektin kanssa yhteistyötä tehnyt prosessoritoimittaja Texas Instruments lähetti koodimuutokset ti.com-sähköpostipalvelimelta. Näiden tahojen koodimuutosehdotukset ja kommentoinnit tilastoitiin omina luokkinaan. Kaikki muut ohjelmakoodin kehittäjät ja kommentoijat ryhmitettiin tilastoinnissa yhdeksi joukoksi.

3.3 Analysoinnissa käytetyt ohjelmat

Tietokannan analysoinnissa käytettiin Python-kielistä ohjelmaa, joka lajittelee halutut tiedot työn tarpeiden mukaisesti. Kyseinen ohjelmointikieli valittiin sen kehitysnopeuden ja laajan kirjastotarjonnan vuoksi. Sähköpostien lataamiseksi gmane.org-tietokannasta käytettiin yksinkertaista Python-ohjelmaa (liite 3), joka hakee kaikki sähköpostit valittujen päivämäärien väliltä. Näitä arkistoja käsiteltiin Python-kielisillä ohjelmilla, jotka suunniteltiin tätä tarkoitusta varten (liite 2).

Linuxin kerneliin kohdistuvat muutokset tulisi lähettää muotoilemattomana tekstinä. Tästä huolimatta osa suunnittelijoista lähettää muutosehdotukset sähköpostin liitteenä. Tämä aiheuttaa ylimääräistä työtä niin katselmoijille kuin tässä työssä käytetyille ohjelmille. Sähköposteja analysoiva ohjelma laventaa liitteen osaksi alkuperäisen sähköpostin runkoa ja etsii tunnisteita tästä liitteet sisältävästä postista.

Suunniteltu Python-ohjelma käyttää seuraavia RFC 2822:n määrittelemiä sähköpostin kenttiä haluttujen tietojen luokittelemiseksi:

Message-ID Yksilöllinen tunniste sähköpostille, kyseisen viestin ID

In reply to Alkuperäisen viestin ID mikäli tutkittava viesti on vastaus

References Lista niiden sähköpostien tunnistimista, joihin tämä viesti on vastaus

From Viestin alkuperäinen lähettäjä ja lähettäjän verkkotunnus

Koodimuutosviesti identifioitiin ensisijaisesti sen sisältämän SHA-1-tunnisteen avulla. Koodin alun tunnistimena käytettiin GNU Patch[40] -yhteensopivaa "+++" -merkkijonoa, jollaisen myös GIT-versionhallintaohjelman "git-format-patch"-komento tuottaa. Muutoksen kohteena olevan ohjelmakoodin rivimäärä lasketaan koodimuutoksen sisältävästä sähköpostista. Tutkittavassa sähköpostitiedostossa lähetetyn ohjelmakoodin loppu tunnistetaan GIT-ohjelman tuottamasta git-versionumerosta,

aiemmassa esimerkissä 1.6.0.4. Tämä tunniste on aina muotoa a.b.c.d jossa a, b, c ja d ovat käytetyn GIT-versionhallintaohjelman versionumeroita.

Postituslistoja käsittelevälle Python-ohjelmalle asetettiin seuraavat vaatimukset:

- Ohjelman on tunnistettava muutosehdotuksen sisältävä sähköposti GNU Patch -tunnisteiden ja muutosehdotuksen identifioivan SHA-1-tarkisteen mukaan. Ohjelman on tunnistettava myös koodimuutosehdotukset, jotka eivät sisällä SHA-1-tarkistetta.
- Ohjelman on tunnistettava SHA-1-tarkisteen avulla onko kyseessä jo aiemmin käsitellyn postituksen duplikaatti ja tässä tapauksessa yhdistettävä säikeet.
- Ohjelman on pystyttävä tunnistamaan kaikki vastaukset koodimuutoksen sisältävään sähköpostiin. Tässä käytetään viestin "Message-ID"- ja "In reply to"- kenttää.
- Ohjelman tulee laskea koodimuutosrivien lukumäärä ja näihin riveihin kohdistuneet katselmointikerrat.
- Ohjelman tulee osata tilastoida omiksi ryhmikseen eri verkkotunnuksista lähetetyt sähköpostit.
- Edellä mainituilta verkkotunnuksilta tulleille koodimuutoksille lasketaan
    - koodimuutosten lukumäärä
    - koodimuutosten yhteispituus
    - koodimuutoksiin kohdistuneet kommentit
    - linux-omap-puuhun hyväksytyt koodimuutokset
    - kommentoitujen koodimuutosten määrä
    - se, kuinka suuri osa kommenteista on tullut samasta verkkotunnuksesta
    - summa kaikkien koodimuutosten saamien koodirivien ja niihin kohdistuvien kommenttien tulolle.
- Ohjelman tulee tilastoida linux-omap-listalle kontribuoivat kehittäjät verkkotunnuksien mukaan.
- Ohjelman tulee laskea jokaiselle päivälle tullee koodimuutosehdotukset.
- Ohjelman tulee laskea viestiketjut, jotka eivät sisällä koodimuutosehdotuksia.

- Ohjelman tulee laskea viestit, joista se ei löydä RFC 2822:n mukaista Message-ID-kenttää.

3.4 Analysoinnin tuottama tieto

Sähköpostit lähetettiin kolmesta projektin kannalta erilaisesta lähteestä, projektihenkilöstöltä (Nokia), prosessoriympäristön toimittajalta (Texas Instruments) ja muilta. Nämä lähteet tunnistettiin verkkotunnuksien nokia.com- ja ti.com perusteella.

Tulleet muutosehdotukset jaettiin sähköpostin lähettäjän organisaation mukaan. Omaksi ryhmäkseen erotettiin nokia.com, ti.com ja muut. Näistä nokia.com-sähköpostiosoitetta käytti Nokia Oyj:n projektiin osallistunut henkilökunta. Tuloksiin tehtiin jakauma siitä, kuinka moneen ohjelmariviin muutokset kohdistuivat. Lisäksi tehtiin histogrammi, jossa kuvataan minkä kokoisia koodimuutoksia postituslistalle lähetettiin. Kommenttien määrä esitettiin erikseen projektin henkilöstön, TI:n ja muiden tekemiin muutoksiin.

Yhteisöltä saadun katselmointiavun kokonaismäärää kuvattiin yhdellä luvulla, joka saatiin seuraavasti:

$$A = \Sigma(Pl * Nc)$$

Jossa Pl = kommentoitavan koodimuutoksen koko ohjelmariveinä ja Nc siihen kohdistettujen kommenttien lukumäärä.

Tässä työssä kaikki tieto koottiin yhteen eikä tutkittu erikseen eri sähköpostilistojen käyttäytymistä. Tämä ratkaisu tehtiin koska kehittäjät lähettävät tekemänsä ohjelmakoodin muutosehdotukset sekä omap-linux-postituslistalle, että kyseessä olevaan liityntään erikoistuneelle listalle.

Kaikki sähköpostilistalla olleet viestit, jotka eivät sisältäneet koodimuutoksia, käsiteltiin tilastollisella otannalla. Niistä tutkittiin sitä, kuinka suuri osa viesteistä käsitteli projektin kannalta oleellisia ongelmia. Tällaiset viestit sisältävät tyypillisesti joko Linuxia tai käytettyä prosessoriympäristöä koskevia kysymyksiä.

Näytteenotossa käytettiin +-5% luottamusväliä ja 95% todennäköisyyttä. Koska etsitty parametri, onko kyseessä hyödyllinen viestiketju, voi saada vain kaksi diskreettiä arvoa (hyödyllinen, hyödytön), voidaan käyttää suoraan binomijakauman näytteenoton koon kaavaa[41].

$$n = \frac{z_{\alpha/2}^2}{4d^2}$$

arvot sijoitettuna, z binomijakauman taulukosta[Binomijakauma] (95%) = 1,96, d = 5% = 0,05

Tarvittavan näytteen koko

$$n = \frac{1{,}96^2}{4 \times 0{,}05^2} = 384{,}16$$

Käytännössä tutkitut sähköpostit jaettiin neljään projektin kannalta hyödylliseen luokkaan, OMAP-prosessoriympäristöä käsittelevät, Linuxiin ja sen kerneliin liittyvät, GIT-versionhallintaan liittyvät ja N900-laitetta käsittelevät kysymykset. Mihinkään näihin luokkiin kuulumattomat katsottiin projektin kannalta merkityksettömiksi. Tilastollinen otos tehtiin ainoastaan linux-omap-listasta. Yleiset keskustelut linux-mmc- ja linux-usb-listalla ovat spesifisiä kyseisille teknologia-alueille, eikä niiden katsottu tuottaneen projektille siinä määrin hyötyä, että olisi ollut mielekästä tämän työn yhteydessä analysoida niitä tarkemmin. Muita kuin koodimuutoksia koskevien sähköpostien tarkastelu rajattiin siis koskemaan vain linux-omap-listalla olleita viestejä.

Ne viestit, joita ohjelma ei tunnistanut, tilastoitiin omana ryhmänään. Tunnistamatta jäivät viestit, joista ei löydetty Message-ID-kenttää. Mikäli näiden viestien määrä nousee korkeaksi, on syy ongelmaan tutkittava. Asetin erillisen tutkinnan rajaksi yhden prosentin kaikista käsitellyistä viesteistä.

4. Tulokset

Tässä luvussa esitetään sähköpostilistojen analyyseissa saadut tulokset. Analyyseissa käytettiin Python-kielistä ohjelmaa (liite 2).

4.1. Sähköpostikirjaston tunnusluvut koodimuutoksille

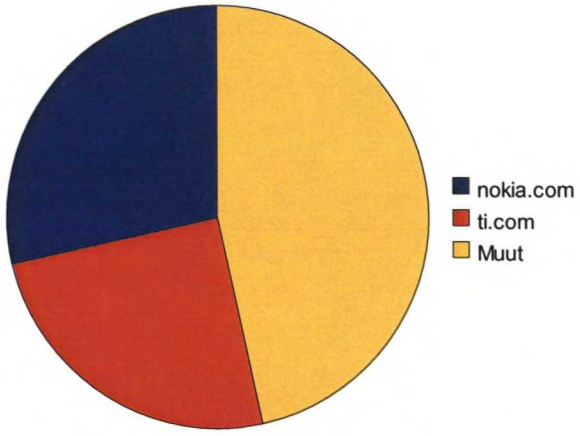
Tarkastelutiedosto: linux-omap-sähköpostilista.

Tarkastelujakso 25.3.2008 - 17.11.2009, yhteensä 603 päivää.

Taulukko 1: linux-omap-postituslistan tilastot.

Parametri	Lukumäärä /kappaletta
sähköposteja tarkastelujaksolta yhteensä	19598
viestiketjua yhteensä	7114
koodimuutoksi sisältäviä sähköpostiketjua	4410
viestiketjua muita kuin koodimuutoksia käsitteleviä asioita	2704
virheelliseksi tulkittuja sähköpostiviestejä, jotka eivät noudattaneet RFC 2822:a	46
koodimuutoksien yhteenlaskettu rivimäärä	1134420
koodimuutosehdotusta nokia.com:sta	1293
koodirivejä nokia.com:sta	313572
kommenttia edellisiin	1649
kommenttia nokia.com:in sisältä	507
koodimuutosta nokia.com:sta jota ei ole kommentoitu	1081
koodimuutosehdotusta ti.com:sta	867
koodirivejä ti.com:sta	279527
kommenttia edellisiin	1542
kommenttia ti.com:in sisältä	590
koodimuutosehdotusta muualta kuin nokia.com:sta tai ti.com:sta	2250
koodirivejä muualta kuin nokia.com:sta tai ti.com:sta	541321

Kuvassa 14 on esitetty jakauma ohjelmakoodin alkuperästä. Projektin ulkopuoliset tahot tuottivat lähes puolet (47,7%) kaikesta laitteen kernelissä käytetystä ohjelmakoodista.



Kuva 14. Tuotteen kernelin ohjelmakoodin alkuperän mukaan

Koodimuutosehdotuksia sisältäviä sähköposteja, joista löytyi OMAP1- tai OMAP4-määrite, laskettiin 702 kappaletta. Tämä vastaa 3.6% kaikista koodimuutosehdotuksista. Näitä ei laskettu mukaan muihin tilastoihin.

Koodimuutoskatselmointien laskennallinen hyöty nokia.com- ja ti.com:sta tulleille koodimuutosehdotuksille:

$$A = \Sigma(Pl * Nc)$$

Jossa Pl = kommentoitavan koodimuutoksen koko ohjelmariveinä ja Nc siihen kohdistettujen kommenttien lukumäärä.

A = 313447 nokia.com koodimuutoksille, ja A = 548809 ti.com koodimuutoksille. Tätä tunnuslukua ei laskettu muiden kuin nokia.com:in ja ti.com:in muutosehdotuksille

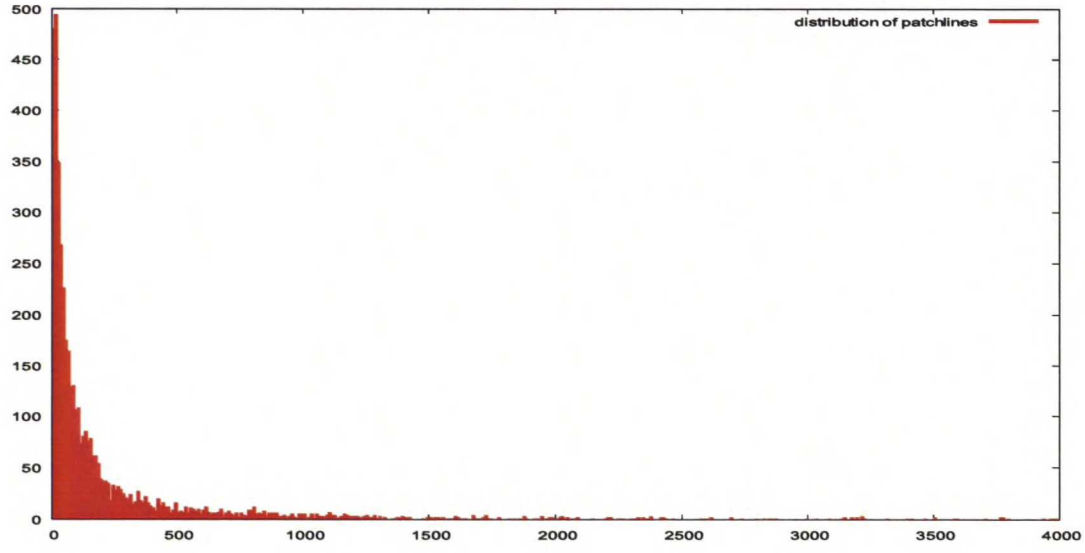
Taulukossa 2 on esietty tilastotietoja muutosehdotuksista. Maksimipituudet edustavat hyvin poikkeuksellisia koodimuutoksia, tyypillinen pituus koodimuutokselle on alle 100 riviä. Taulukkoon 3 on kerätty tietoja kommentoinneista. Noin kolmasosa projektin saamista kommenteista tuli projektihenkilökunnalta. Valtaosa koodimuutoksista sai vain yhden julkisen kommentin.

Taulukko 2: Muutosehdotuksen alkuperän mukaan lasketut tilastot.

Sähköposti-palvelin	Muutos-ehdotuksia /kappaletta	Minimi-pituus /riviä	Pituuksien mediaani /riviä	Pituuksien keskiarvo /riviä	Maksimi-pituus /riviä
nokia.com	1293	6	67	242	8754
ti.com	867	6	93	288	4780
muu	2250	6	81	227	4243
kaikki	4410	6	79	225	8754

Taulukko 3: Muutosehdotuksen alkuperän ja kommentointi.

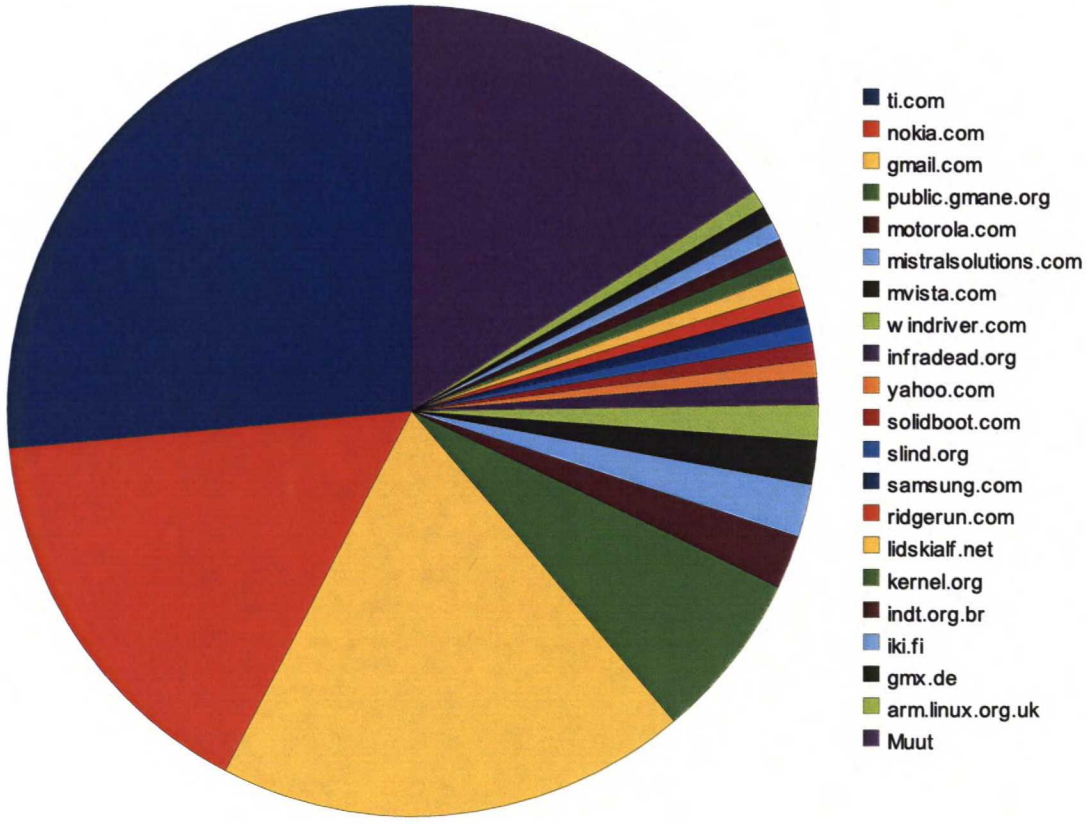
Sähköposti-palvelin	Muutos-ehdotuksia	Kommentteja /kappletta	Kommentteja samasta palvelimesta /kappaletta	Kommentteja muualta /kappaletta
nokia.com	1293	1649	507	1142
ti.com	867	1542	590	952
muu	2250	2829	n/a	2829



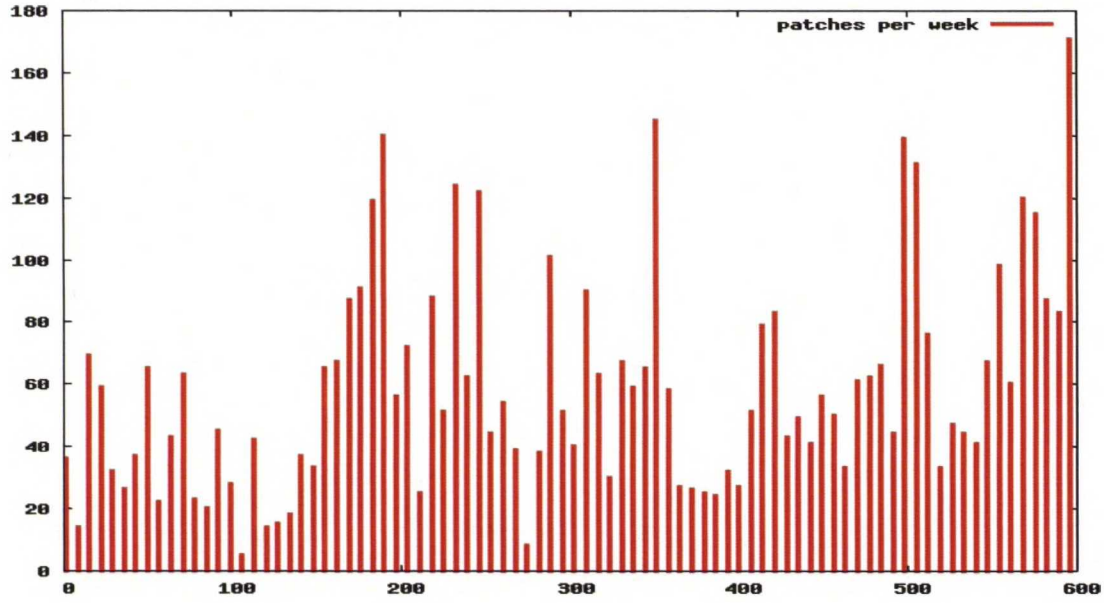
Kuva 15. Koodimuutosehdotusten jakauma niiden koon mukaan. Muutokset on kvantisoitu 10 rivin tarkkuuteen.

Histogrammi kuvassa 15 kuvaa lähetettyjen koodimuutosehdotusten kokojen hajontaa. Valtaosa koodimuutoksista oli alle 100 rivin kokoisia (koon mediaani 79 riviä). Yli 4000 rivin muutokset on jätetty kuvasta pois niiden pienen määrän vuoksi.

Koodimuutoksia tarkasteltuun kaupalliseen projektiin toimitti 289 eri suunnittelijaa. Kuva 15 esittää näiden jakauman sähköpostipalvelimen mukaan. Eniten muutosehdotuksia lähettäneitä henkilöitä oli prosessoriympäristön toimittajalla, 57 kappaletta. Kolmanneksi eniten projektilla, 45 kappaletta. Valtaosa ohjelmakoodia kirjoittaneista henkilöistä ei ollut työsuhteessa kummankaan edellisen kanssa. Joukossa on muutama kaupallinen yritys, mutta merkittävä osa listassa olevista sähköpostipalvelimista on julkisia, jonne kuka tahansa voi luoda käyttäjätunnuksen.



Kuvassa 16. Projektille ohjelmakoodia tuottaneiden suunnittelijoiden sähköpostipalvelimien jakauma.



Kuva 17. Histogrammi koodimuutosten ajallista jakaumaa tarkastelujaksolle. Histogrammin vasen laita on tarkastelujakson alku, 25.3.2008 ja oikea laita on tarkastelujakson loppu, 17.11.2009.

Ohjelmakoodia tuotettiin koko projektin ajan (kuva 17). Koodin julkaiseminen painottui jonkin verran projektin loppupuolelle.

4.2 Koodimuutosehdotusten koko ja alkuperä

Yhteensä koodimuutoksia tarkasteltuun kaupalliseen projektiin tuotti 289 eri ohjelmoijaa. Sähköpostilistalle tulleet koodimuutosehdotukset jaettiin niiden alkuperän mukaan kolmeen luokkaan.

- Projektin henkilöstöltä tullut ohjelmakoodi, joka tunnistettiin sähköpostien lähettäjän verkkotunnuksen mukaan, joka tässä tapauksessa oli nokia.com.
- Prosessoriympäristön toimittajan kontribuoima ohjelmakoodi. Se tunnistettiin ti.com-verkkotunnuksesta.
- Muutosehdotukset, joita koskevat sähköpostit lähetettiin muista kuin edellä mainituista verkkotunnisteista.

Koodimuutosten ajallisesta hajonnasta (kuva17) voitiin todeta, että OMAP3:lle oli lähetetty koodimuutoksia koko tarkastelujakson ajan. Lähetettyjen koodimuutosten määrä kasvoi projektin loppua kohden.

Suurin osa koodimuutoksista on kooltaan pieniä ja helposti kommentoitavia. Uuden ohjelmakoodin joukossa on muutama hyvin iso kokonaisuus. Tällainen on esimerkiksi OMAP3:n energiankäytön hallinnasta vastaava ohjelmisto.

Projektin ulkopuoliset kehittäjät voivat olla kolmansien osapuolten yritysten suunnittelijoita, jotka kehittävät kyseistä ajuria oman yrityksensä tarpeisiin, tai muita OMAP3-prosessoriympäristöstä kiinnostuneita Linux-yhteisön jäseniä. Heidän kiinnostustansa voi perustella kyseiseen prosessoriympäristöön tarjolla olevalla edullisella "BeagleBoard"-kehitysympäristöllä[42].

Sähköpostilistalle lähetetystä ohjelmakoodista varsinaiselta projektihenkilöstöltä oli peräisin 27,7% ja projektin kanssa kiinteässä yhteistyössä toimineelta Texas Instruments:lta 24,6%. Yhteensä tämä on 52,3% ja vastaa noin puolta kerneliin tarkastettua projektia varten tehdystä ohjelmakoodista. Projektin- ja siihen kiinteässä yhteistyössä olevien tahojen ulkopuolelta tulevan koodin määrä, 47,7% kaikesta uudesta ohjelmakoodista, on projektin kannalta merkittävä. Ulkopuoliset tahot tuottivat koodia pääasiassa geneerisille laitteille ja standardiliitynnöille, jotka löytyvät julkisesti saatavilla olevasta BeagleBoard-kehityslaitteesta.

4.3 Linux-yhteisöltä saatu ohjelmakoodi

Koodimuutoksia ja katselmointikommentteja tehneiden suunnittelijoiden alkuperä tunnistettiin heidän lähettämiensä sähköpostien verkkotunnuksien avulla. Tarkastelujaksolta löydettiin lähetettyjä koodimuutosehdotuksia yhteensä 4410 kappaletta. Nämä sisälsivät yhteensä 1134420 riviä ohjelmakoodia. Normaali teollisuusohjelmiston tuottavuus on keskimäärin 400 riviä valmista, toimivaa ohjelmakoodia kuukaudessa[43]. Muutosehdotuksissa ollut koodimäärä vastaa tässä tapauksessa 2836 miestyökuukauden työmäärää. Tarkasteltu ajanjakso oli noin 20 kuukautta jolloin voidaan todeta ohjelmakoodin tuottamiseksi vaaditun 142 ohjelmoijaa.

- Nokia.com:sta tulleet ohjelmistomuutokset olivat kooltaan yhteensä 313572 riviä, joka vastaa 783 miestyökuukautta ja merkitsee tarkastelujaksolla keskimäärin 39 ohjelmoijan työpanosta.
- Texas Instrumentsin lähettämä koodimäärä, 279527 riviä, vastaa 698 miestyökuukautta ja tarkasteluaikana 35 hengen työpanosta.
- Näiden tahojen ulkopuolelta projektin hyödyntämää ohjelmakoodia saatiin 541321 riviä. Tämä vastaa 1353 miestyökuukautta ja projektin 20 kuukauden

kestolle jaettuna 68 ohjelmoijaa. Tämä hyöty oli projektille hyvin merkittävää.

Tässä arvioidut työntekijöiden määrät sopivat hyvin yhteen havaittujen koodimuutosten tekijöiden määrän kanssa. Taulukosta 5 nähdään että nokia.com:sta muutoksia lähetti nokia.com:sta 45 tekijää, ti.com:sta 77 tekijää ja muualta 167 tekijää.

4.4 Ohjelmakoodin katselmointi

Ennen työn aloittamista oletuksena oli että suurin etu tiiviistä yhteistyöstä Linux-yhteisön kanssa saavutettaisiin ohjelmakoodien julkaisuun liittyvällä katselmoinnilla. Boehm toteaa[44] että 60 prosenttia järjestelmästä löydetyistä ohjelmointivirheistä voidaan löytää koodikatselmoinneissa. Perinteisen funktionaalisen testaamisen avulla saavutetaan vain 30-50% taso[45]. Laadukkaan koodikatselmoinnin arvo projektille on helposti ymmärrettävissä.

Linux-kernelin tapauksessa katselmoinnin arvoa lisää vielä se, että katselmoinnin tekevät kunkin alueen erikoisasiantuntijat, joiden palkkaaminen projektiin vastaaviin tehtäviin olisi monessa tapauksessa vaikeaa tai mahdotonta. Tässä tutkimuksessa katselmoinnin yhteisvaikutus laskettiin kertomalla katselmoitujen ohjelmarivien määrä katselmointikertojen määrällä. Näin menettelemällä saatiin suurimmassa osassa tapauksista riittävän tarkka kuva todellisesta hyödystä.

Ohjelmamuutoksien kommentointi on mielekästä käsitellä kolmessa ryhmässä. Ensinnäkin projektihenkilöstön lähettämien koodimuutosehdotuksien kommentointi. nokia.com-verkkotunnuksesta lähetettyjä koodimuutosehdotuksia oli yhtensä 1293 kappaletta ja näissä oli muutoksen alaista ohjelmakoodia 313572 riviä. Näistä muutosehdotuksista katselmointikommentteja sai alle 20%, 212 kappaletta.

Kommenttien määräksi laskettiin sivun 36 kaavan mukaan 313447. Kaavalla lasketaan summa sille, kuinka monta katselmointikertaa kuhunkin ohjelmariviin kohdistui. Tämä lukema kuvaa sitä, kuinka monta riviä yksittäisen katselmoijan olisi pitänyt tutkia, tuottaakseen vastaavan hyödyn projektille. Lisäksi on huomattava, että tässä katselmoinnin suorittivat kunkin aihealueen erityisosaajat. Kattavan katselmoinnin järjestäminen ohjelmakoodille on hyvin hankalaa ja aikaa vievää[46]. Kullekin osa-alueelle on löydettävä kyseisen alueen asiantuntijat ja sovitettava aikataulut katselmointikokouksen järjestämiseksi. Lisäksi jokaisen katselmoijan on tutustuttava katselmoitavaan ohjelmakoodiin ja annettava palaute. Käytettäessä Linuxin kernelkehitysmallia, kukin asiantuntija seuraa automaattisesti oman alueensa kehitystä ja voi antaa palautetta koodimuutoksista sitä mukaa kuin niitä julkaistaan. Menetelmä on tehokas, se

sallii niin yrityksen sisäisten kuin ulkoistenkin asiantuntijoiden osallistumisen. Koodin katselmoinnissa arvioin sopivaksi kerralla käsiteltäväksi koodimääräksi 400 riviä. Tähän tulisi kulua aikaa noin 1,5 tuntia. Tämä pohjautuu konsulttiyhtiö SmartBear Inc:in tekemään tutkimukseen[47]. Näin arvioimalla projektin saamaksi hyödyksi voidaan laskea 784 katselmointikertaa, joista jokainen kestäisi 1,5 tuntia. Suoraan 7,5-tuntisiksi työpäiviksi muutettuna hyöty olisi 156 työpäivän työpanos. Tämä vastaa yhden henkilön lähes kahdeksan kuukauden työpanosta. Tässä tapauksessa on kuitenkin huomioitava, että katselmoinnin ovat suorittaneet henkilöt, joiden saaminen tekemään vastaavaa työtä yrityksen palvelukseen, olisi ollut erittäin vaikeaa, tai jopa mahdotonta. Katselmoinnin hyöty tulisi tarkastella laajemmin, mittaamalla sen merkitystä ohjelmakoodin laadulle. Tämän voisi tehdä vertaamalla katselmoitua koodia katselmoimattomaan ja laskemalla näistä raportoitujen ohjelmavirheiden määrä.

Huomionarvoinen seikka on se, että läheskään kaikki katselmoijat eivät kommentoi muutosehdotusta, mikäli siinä ei ole mitään huomautettavaa, tai jos joku toinen on jo huomioinut saman asian. Tästä johtuen analysointituloksista laskettu katselmointimäärä on todennäköisesti huomattavasti pienempi kuin todellinen.

4.5 Tunnusluvut muille viesteille

Kommunikaation paranemisen aikaansaamaa etua projektille on hankala mitata kvantitatiivisesti. Linux-omap-postituslistalla on käyty paljon keskustelua arkkitehtuurisista ratkaisuista ja haettu vastauksia teknisiin kysymyksiin. Aihealueen ohjelmakoodin muutoksia käsittelevältä postituslistalta saa aihetta käsitteleviin kysymyksiin asiantuntevan vastauksen nopeasti. Listaa seuraavista sadoista asiantuntijoista löytyy lähes poikkeuksetta henkilö, joka tietää vastauksen vaikeimpiinkin kysymyksiin. Näihin ongelmiin ovat vastanneet sekä N900-projektiin osallistuneet henkilöt, että Linux-asiantuntijat projektin ulkopuolelta. Teknisiä ongelmia käsittelevät keskusteluketjut ovat projektin kannalta tärkeitä ja ne lisäävät oleellisesti sen Linux-yhteistyöstä saamaa hyötyä. Postituslistalta löytyneiden teknisten keskustelujen määrä löytyy taulukosta 6. Yhteenlaskettu teknisten keskustelujen sähköpostimäärä oli 2704, joista yhden prosentin arvioitiin olevan muita kuin projektia hyödyttäviä keskusteluja. Tämä tarkoittaa että projektia hyödyttäviä teknisiä keskusteluja käytiin linux-omap-listalla 4-5 sähköpostin verran joka päivä. Tämä keskustelu käytiin pääasiassa projektin ulkopuolisten henkilöiden kesken. Tästä voidaan katsoa olleen projektille mainittavaa hyötyä.

Projektissa mukana olleet henkilöt osallistuivat Linux-kernelin kehitykseen samalla

tavalla kuin muutkin Linux-yhteisön kernelkehittäjät. Näin menetellessään he tutustuivat oman vastuualueensa ulkopuolisiin teknisiin toteutuksiin ja näiden Linux-toteutuksen arkkitehtuuriin ja implementaatioon. Saavuttaessaan laajemman ymmärryksen koko Linuxin kernelin arkkitehtuurista ja sen osien implementoinnista, on suunnittelijan helpompi valita oikeat ratkaisut oman vastuualueensa ohjelmiin ja saada ne toimimaan optimaalisesti yhteen muiden osien kanssa. Projektin ja yrityksen on myös mahdollista hyvin lyhyen tutustumisjakson jälkeen käyttää tällaista suunnittelijaa muidenkin alueiden vastuullisena suunnittelijana.

Taulukko 6: Luvussa kaksi kuvatun otannan tulos niiden sähköpostien sisällöstä, jotka eivät sisältäneet koodimuutoksia. Tällaiset viestit käsiteltiin otantaperiaatteella, jonka perusteet on esitetty luvussa 3.4.

Aihealue	Viestien määrä/kappletta	Prosenttiosuus
OMAP prosessoriympäristö	205	53,4%
Linux kernel	74	19,3%
GIT versionhallinta	6	1,5%
N900 spesifiset	7	1,8%
Muut oleelliset viestit	88	22,9%
Projektille merkityksettömät	4	1,0%
Summa	384	100,0%

On oletettavaa, että yritys saa tällaisessa projektissa hyviä kontakteja avoimen koodin ohjelmistokehittäjiin. Nämä kontaktit saattavat johtaa sellaisten henkilöiden palkkaamiseen, joiden löytäminen normaalien rekrytointikanavien kautta olisi hankalaa. Samaten yrityksen julkinen osallistuminen Linux-kernelin kehittämiseen tuo yritykselle positiivista julkisuutta avoimen koodin kehittäjien keskuudessa.

4.6 Hylätyt sähköpostit

Postituslistalta löydettiin 46 sellaista sähköpostia, joissa ei ollut niitä RFC 2822:n määrittelemiä identifiointikenttiä, joita analyysiohjelma etsi, tai vaihtoehtoisesti ohjelma ei pystynyt tulkitsemaan viestin sisältöä. Tällaisten sähköpostien määrä oli kokonaismäärästä 0,23%. Totesin tällaisten sähköpostien määrän merkityksettömän pieneksi, enkä tutkinut vikojen syitä. Tällaiset sähköpostit jäivät kaikkien tilastojen ulkopuolelle.

5. Johtopäätökset

Yhteenvetona tästä tutkimuksesta voidaan todeta seuraavaa:

- Linux-yhteisöltä saatua ohjelmakoodia oli paljon, ja tästä saatu hyöty on ollut tutkitulle projektille erittäin merkittävä. Ohjelmakoodina saadun hyödyn lasketaan vastaavan yhteensä 1353 miestyökuukautta. Tämä vastaa 68 ohjelmoijan kokopäiväistä työpanosta koko projektin tarkastelujakson ajan.
- Linux-yhteisöltä saatu katselmointityö oli laajaa ja siitä saavutettu hyöty projektille merkittävää. Yhteensä arvioitu 156 työpäivää.
- Avoimen koodin kehittämisessä käytetyn sähköpostilistan käyttö teknisten kysymysten ratkaisemiseen oli laajaa ja siitä saatu hyöty projektille merkittävää. Tämän hyödyn tarkempi arviointi jäi tässä työssä selvittämättä.

Tässä työssä selvitettiin niitä hyötyjä, joita mobiililaiteprojekti saa toimiessaan kehitysaikana tiiviissä yhteistyössä Linux-yhteisön kanssa. Työn tulokset osoittavat kiistatta sen, että yhteistyö Linux-yhteisön kanssa toimi tehokkaasti ja oli tuottavaa. Merkittävää osaa projektin kernelmuutoksista kommentoitiin ja projekti sai Linuxin kerneliä kehittävältä yhteisöltä runsaasti käyttökelpoista ohjelmakoodia. Projektin aikana sen henkilöstö oppi tuntemaan Linux-yhteisön ja kernelin kehitystyön vaatimukset. Kaikki kontribuutiot Linux-kerneliin lähetettiin kehittäjän nimellä, näin projektiin osallistuneet suunnittelijat tulivat tunnetuksi Linuxin kernelyhteisössä. Yhteisössä tunnetuksi tuleminen on sujuvan kommunikaation kannalta ensiarvoisen tärkeää. Koodien katselmointi sujuu nopeammin ja kommentoijien luottamus omaan osaamiseensa kasvaa jokaisen hyväksytyn patchin myötä. Seuratessaan oman alueensa postituslistaa, kehittäjien ammattitaito ja oman erikoisalueensa tuntemus kasvavat. Samaan aikaan projektissa työskentelevät kehittäjät verkottuvat muiden kernelkehittäjien kanssa. Postituslistoista tuli merkittävä teknisten keskustelujen foorumi. Avoimen ohjelmakoodin kehitysmalli sallii myös joustavasti muiden kuin omien vastuualueiden kehityksen seuraamisen ja näihin tutustumisen. Alkuperäisenä tarkoituksena oli tutkia useille postituslistoille lähetettyjä ohjelmakoodeja. Työn kuluessa havaittiin että lähes kaikki koodi oli lähetetty linux-omap-listalle.

6. Työn tulosten tarkastelu

Tämä diplomityö rajattiin niin, että se tutki yhtä projektia ja sen Linux-yhteistyöstä saamaa hyötyä. Tutkimuksen rajaus oli tiukka sallien suhteellisen syvälle luotaavan analyysin. Samalla valittu rajaus jätti useita mielenkiintoisia asioita tämän tutkimuksen ulkopuolelle. Työ saatiin tehtyä määriteltyjen rajojen puitteissa ja tulokset kuvasivat niitä asioita, joita tutkimuksessa haettiin.

Työssä saadut tulokset vastasivat varsin hyvin ennakko-odotuksia. Selvä poikkeus oli suurehko Linux-yhteisöltä tulleen ohjelmakoodin määrä. Toinen odottamaton seikka oli asiaan kuulumattoman keskustelun lähes täydellinen puuttuminen linux-omap-palstalla. Tämä havaitaan taulukon 6 tuloksista, jossa on käyty läpi niitä sähköposteja, jotka eivät sisältäneet koodimuutoksia. Vaikka tässä työssä tarkasteltiinkin Linux-yhteistyön merkitystä projektin kannalta, saatiin sivussa paljon viitteitä yhteistyön hyödyllisyydestä niin yksityiselle ohjelmankehittäjälle kuin projektin omistavalle yritykselleKin. Sähköpostikeskusteluista näkyy, kuinka ohjelmakoodejaan sinne lähettävät ohjelmoijat pääsevät ajan kuluessa yhä paremmin sisään Linux-yhteisöön. Linux-yhteisön tuntemisesta ja sen kehitystapojen oppimisesta, on selvää hyötyä ohjelmoijalle. Mahdollisten ongelmatapausten selvittäminen on helpompaa, kun tarjolla on yksinkertainen kanava ratkaisujen hakuun. Lähettäessään ohjelmakoodeja omalla nimellään Linuxin kerneliin, saavat ohjelmoijat nimensä tunnetuksi ja heidän itseluottamuksensa kasvaa jokaisen hyväksytyn koodimuutoksen myötä. Ohjelmoijien kompetenssi kasvaa ja tiukat koodaussäännöt tekevät ohjelmoinnista kurinalaista. Samalla toisten koodien katselmoinnista ja kommentoinnista tulee luonnollinen osa kehitystyötä.

Aiemmassa luvussa kuvattujen etujen lisäksi tulee Linuxin käytöstä projektille muitakin hyötyjä. Mikäli tuotetta valmistetaan suuria määriä, saavutetaan käyttöjärjestelmän lisenssimaksujen poistumisella merkittävä taloudellinen etu. Avoimen koodin käyttö saattaa myös mahdollistaa muiden yritysten ja projektien aiemmin tekemän ohjelmistokannan hyödyntämisen. Samaten yritys voi suhteellisen kevyellä sopimusmenettelyllä tehdä yhteistyötä muiden samaa prosessoriympäristöä käyttävien valmistajien kanssa. Tällaisestahan on selkeä esimerkki Apache-projektissa. On kuitenkin huomattava että Linuxin käyttö sinänsä ei vaadi aktiivista yhteistyötä Linux-yhteisön kanssa.

6.1 Lähdeaineistosta

Avoimeen Linux-lähdekoodiin perustuvaa kernelkehitystä on mahdollista arvioida pelkästään julkisista lähteistä saatavan tiedon perusteella. Tässä työssä arviot yhteistyön hyödystä tehtiin analysoimalla yksinomaan vapaasti saatavilla olevia sähköpostiarkistoja. Työn tulosten tarkastelu on tällöin jonkin verran ongelmallista. Yrityksen sisäiset käytännöt ja niistä aiheutuvat kustannukset on pitänyt jättää kokonaan tämän työn ulkopuolelle. Julkisesti saatavissa oleville tiedoille ei ole myöskään mahdollista saada vertailukohtaa niistä yrityksen projekteista, jotka on tehty käyttämällä suljetun koodin ohjelmistoja.

On ymmärrettävää, ettei tässä työssä käsitellystä aiheesta ole aiemmin tehty julkista tutkimusta, sillä kattava ja vertailevaan aineistoon perustuva tutkimus, voidaan tehdä vain käyttämällä hyväksi yrityksen sisäisiä, luottamuksellisia tietoja. Mikäli tavoitteena olisi saada tarkka kuva kaikista positiivisista ja negatiivisista vaikutuksista mitä avoimen koodin käyttö tuo yritykselle, olisi ensiarvoisen tärkeää pystyä vertaamaan avoimen koodin ohjelmistoprojektia vastaavaan suljettuun ohjelmakoodiin perustuvaan projektiin. Tällöin voitaisiin huomioida kaikki asiaan vaikuttavat tekijät. Näitä tekijöitä ovat muun muassa yrityskulttuuri ja projektiin osallistuvien tekijöiden keskimääräinen kompetenssi. Myös uusien asioiden ja menettelytapojen oppimiseen kuluva aika tulisi kyetä arvioimaan. Vertailussa olisi myös hyvä ottaa mukaan useampi projekti. Yksittäisen laitteen tuotekehityksen yhteydessä projektin kestoon ja tuottavuuteen vaikuttaa suuri määrä tämän tutkimuksen aiheesta riippumattomia tekijöitä.

Ensimmäisessä vaiheessa tutkimuksen voisi koskea käsittämään useita avoimen ohjelmakoodin projekteja, joissa yritykset ovat valmistaneet kaupallisia tuotteita. Tässä olisi kuitenkin edelleen ongelmana kaupallisen yrityksen sisäisen vikatietokannan puuttuminen analyyseista. Toisessa vaiheessa mukaan voisi pyrkiä ottamaan vertailukohdaksi suljetun koodin projekteja, jolloin avoimen ohjelmakoodin, ja linux-yhteisön, merkitys selviäisi tyhjentävämmin. Tämä edellyttäisi, että tarkasteltavan projektien omistava yritys olisi mukana tutkimuksessa, ja sallisi pääsyn yrityksen sisäisiin tietokantoihin.

6.2. Koodin katselmoinnista

Yritysten käyttämä katselmointikäytäntö on yleensä kaksivaiheinen. Ensin katselmointiin osallistuville jaetaan katselmoinnin kohteena oleva teksti tai ohjelmakoodi. Tämän jälkeen heidän odotetaan tutustuvan katselmoitavaan aineistoon.

Määrätyn ajan kuluttua järjestetään palautekokous, jossa kerätään katselmoinnissa tulleet havainnot ylös ja päätetään mitä muutoksia katselmoituun dokumenttiin tai ohjelmakoodiin tulee tehdä. Kun tutkitaan tässä työssä katselmoinnin tai kommentoinnin kohteena olleita ohjelmakoodeja, huomataan että niitä on ollut 4410 kappaletta (taulukko 1). Mikäli tällainen määrä ohjelmakoodia olisi katselmoitu perinteisellä tavalla, niin että kolme henkilöä olisi tutustunut kaikkeen koodin, olisi työmäärä ollut huomattavasti isompi kuin kappaleessa 4.4 laskettu 156 miestyöpäivää. Jotta saavutettu hyöty voitaisiin analysoida tarkemmin, olisi pystyttävä arvioimaan kuinka monta henkilöä on tutustunut sähköpostilistalle lähetettyyn koodimuutokseen.

Koodien katselmointi sähköpostilistalla ei ole yhtä järjestelmällistä, kuin yrityksen sisällä järjestetyissä katselmoinneissa. Linuxin kernelkehityksessä käytetyssä katselmointitavassa on mahdotonta tietää, kuinka moni henkilö tutustuu koodimuutokseen, jättämättä siitä minkäänlaista kommenttia. Näin voi käydä erityisesti siinä tilanteessa, että koodia on jo kommentoitu, tai silloin kun koodi on katselmoijan mielestä moitteetonta. Analyysissa löydettiin iso joukko koodimuutosehdotuksia, joihin ei ollut tullut yhtään kommenttia ja jotka oli siitä huolimatta otettu osaksi linux-omap-kerneliä. Kysyttäessä alueen ylläpitäjän syytä tähän, hän kertoi itse tarkistaneensa kyseiset koodit ennen niiden integrointia kerneliin. Varsinkin projektin alkuaikoina ylläpitäjä kertomansa mukaan jätti kirjoittamatta sähköpostilistalle tätä hyväksyntää koskevan kommentin.

Työmäärien laskeminen pelkkien sähköpostitilastojen avulla on haastava tehtävä. Aiemmin mainituista syistä johtuen todellinen katselmoijien määrä on saattanut olla merkittävästi suurempi, kuin sähköposteista laskettu. Tuloksissa oleva laskettu määrä on joka tapauksessa minimi katselmointimäärä, joka koodeihin on kohdistunut. Katselmoinnin vaatiman työajan arvioiminen on monella tapaa hankalaa. Virallisissa katselmoinneissa siihen osallistujat yleensä tutustuvat ensin katselmoitavaan ohjelmakoodin kukin omalla tahollaan ja kokoontuvat sovittuun aikaan antamaan palautetta. Itse katselmoinnin ja palautteen antamisen välinen aika voi olla huomattava, pahimmassa tapauksessa viikkoja. On selvää että palaute ei tällaisessa tapauksessa ole yhtä tarkkaa kuin välittömästi annettuna.

Analyysien tuloksista laskemalla saatu 156 työpäivän hyöty on selkeästi alaestimaatti siitä työpanoksesta, jonka projekti on saanut. Lisähyötynä voidaan lisäksi laskea, että Linux-kernelkehityksessä käytettyjen sähköpostilistojen tapaisessa katselmoinnissa palaute tulee asiaan perehtyneiltä ja motivoituneilta katselmoijilta.

6.3 Ohjelmakoodista

Tässä työssä käytetty tarkasteltava aikaväli määriteltiin alkamaan siitä, kun ensimmäinen koodimuutos oli lähetetty linux-omap-listalle. Tarkastelujakso päättyi siihen, kun tuote julkaistiin. Lähetetyn ohjelmakoodin vaatimaa työtä on luonnollisesti tehty arvioitua aloitusajankohtaa aiemmin. On myös todennäköistä, että ennen ensimmäisen koodin julkaisemista, laitteelle on kehitetty usean kuukauden ajan, testaustarkoituksiin käytettyä ohjelmakoodia. Ainakin osa tästä testauskoodista on siirtynyt suoraan, tai hyvin pienillä muutoksilla, varsinaiseksi tuotekoodiksi. Projektin julkaisemisen ottaminen arviointiajan loppukohdaksi, on arviona lähempänä oikeaa. Viimeiset laitteeseen liittyvät ohjelmakoodit on julkaistu postituslistalla 17.11.2009. Tämä on laitteen julkaisua edeltänyt päivä. Laitetta alettiin toimittaa kauppoihin muutaman viikon kuluttua julkaisusta, joten työssä käytetty loppuajankohta, 18.11.2009, on varsin hyvä estimaatti. Edellä kerrotuista syistä johtuen projektin arvioitu kesto kernelohjelmiston tuottamiseen on ollut jonkin verran pidempi kuin tässä työssä käytetyt arviot.

Tutkitulla linux-omap-postituslistalla oli ohjelmakoodia, jota ei käytetty lainkaan N900-laitteessa. Tällaisia olivat joihinkin muihin OMAP3-pohjaisiin toteutuksiin tehdyt laitekohtaiset ohjelmat. Esimerkkinä tällaisesta voidaan mainita BeagleBoard-kehityskortin ledien toimintaa ohjaava ohjelmakoodi. Näiden tarkastellun projektin kannalta irrelevanttien ohjelmakoodien määrä oli kuitenkin vähäinen, eikä niiden huomiotta jättäminen aiheuta oleellista virhettä lopputuloksiin. Mikäli tämänkaltaiset virheet haluttaisiin sulkea kokonaan pois, olisi sähköpostiarkisto käsiteltävä manuaalisesti. Tämä aiheuttaisi huomattavan lisätyön vaikuttamatta kuitenkaan merkittävästi lopputulokseen. Lähes kaikki kerneliin hyväksytty ohjelmakoodi on kuitenkin otettu mukaan valmiin laitteen kerneliin, vaikka kaikelle uudelle toiminnallisuudelle ei ole ollut käyttöä suunnitellussa tuotteessa. On hyvin vaikea arvioida, kuinka suuri osa projektin ulkopuolelta tulleesta ja kerneliin hyväksytystä ohjelmakoodista oli sellaista, jota ei voitu hyödyntää lopullisessa tuotteessa. Lisäksi on huomioitava, että vaikka ohjelmakoodia ei olisi käytetty lopullisessa tuotteessa, on se saattanut olla apuna laitteen ohjelmien maturoinnissa ja auttaa Linux-yhteisön projektia tukeneita suunnittelijoita.

Ulkopuolelta saadun ohjelmakoodin säästämän työpanoksen arviointi on toinen haastava tehtävä. Tässä työssä saatiin projektihenkilökunnan oman työn arvoksi koko projektin keston ajalta 39 ohjelmoijan työpanos. Analysoitaessa nokia.com-sähköposti-palvelimelta linux-omap-listalle lähettäneitä kontribuuttoreita saatiin heidän

lukumääräkseen 45. Tämä vastaa varsin tarkasti sähköpostilistojen avulla tehtyä arviota. Ero tulee siitä, että kaikki ohjelmoijat eivät osallistuneet projektiin sen alusta alkaen. Toinen saatavissa oleva tieto oli ti.com:sta tulleiden koodimuutosten lähettäjien määrä. Analyyseissa laskettiin 77 erillistä kontribuuttoria. Tehdyn työn arvoksi laskettiin 35 henkilöä koko projektin ajalta. Työmääräarviot ovat oikean suuntaisia. Tämä antaa hyvän kuvan laskentamenetelmän luotettavuudesta.

Muualta kuin projektihenkilöstöltä ja prosessorintoimittajalta saatu käyttökelpoinen ohjelmisto vastasi 68 henkilön työpanosta koko projektin tarkastelujakson aikana. Kontribuutioita tuli yhteensä 167 lähettäjältä.

Julkisia lähteitä käyttäen ei voi arvioida sitä, kuinka moni sähköpostipalvelimen osoitteen takana olevista yrityksistä saa projektia hallinnoineelta yritykseltä korvauksen tekemästään työstä. On hyvin mahdollista, että N900-tuoteprojektille tilattiin ohjelmakokonaisuuksia joltain avoimen koodin kehitykseen erikoistuneelta yritykseltä. Tästä johtuen, Linux-yhteisöltä saadun koodin määrä on todellisuudessa luultavasti ollut jonkin verran työn tuloksissa arvioitua pienempi.

6.4. Jatkotutkimuksesta

Tässä työssä on löydetty useita seikkoja, jotka puoltavat tiivistä yhteistyötä kaupallisen tuotekehitysprojektin ja Linux-yhteisön välillä. Työ oli kuitenkin rajattu koskemaan vain yhtä yritystä ja yhtä projektia. Työ on lisäksi tehty analysoimalla julkisesti saatavilla olevia tietoja, tässä tapauksessa kehitystyössä käytettyjä sähköpostilistoja. Tästä työstä puuttuu kokonaan vertailuaineisto muihin samanlaisiin projekteihin, jotka on tehty suljetulla ohjelmakoodilla. Työssä ei ole myöskään vertailua muihin avointa lähdekoodia käyttäviin projekteihin.

Tutkitussa projektissa käytetty kehitystapa ei ole ainoa mahdollinen tapa hyödyntää vapaasti saatavilla olevaa Linux-kerneliä. Julkiselle linux-omap-listalle, ja sieltä Linux-kerneliin mennyttä ohjelmakoodia, voidaan käyttää tuotteessa osallistumatta aktiivisesti Linuxin kernelin kehittämiseen. Tällöin haasteelliseksi tulee kernelversioiden seuraaminen. Mikäli yritys ei itse julkaise tekemiään kernelmuutoksia jo kehityksen aikana, on yrityksen pidettävä huolta, että kerneliin tulevien muutosten vaikutukset huomioidaan kaiken aikaa yrityksen omassa kernelkoodissa. Uusia kernelversioita tulee noin 10 viikon välein, se tarkoittaisi tarkastellun kokoisessa projektissa noin kymmentä koodimuutoskierrosta kaikelle valmiille ohjelmakoodille. Joissakin tapauksissa yrityksen oma kernelversio olisi synkronoitava myös väliversioiden kanssa. Tästä

seuraavan työmäärän arviointi on vaikeaa. Joka tapauksessa se aiheuttaisi kymmenien prosenttien työmäärän kasvun itse kehitetyn ohjelmakoodin osalle. On huomattavasti yksinkertaisempaa integroida oma koodi Linux-kerneliin ja näin varmistaa että kaikki uusi ohjelmakoodi on yhteensopivaa yrityksen kehittämän koodin kanssa. Vertailuaineistoon tulisi kuitenkin mahdollisuuksien mukaan kuulua jonkin tällaisen projektin tunnusluvut.

Tarkkojen työmääräarvioiden saamiseksi tulisi käytettävissä olla yrityskohtaiset tiedot työn tuottavuudesta. Ensiarvoisen tärkeää olisi myös kerätä tiedot siitä, kuinka paljon virheitä kustakin ohjelmakoodin osuudesta löydetään tuotteen julkaisemisen jälkeen. Mikäli avoimen koodin käytöstä, ja Linux-yhteisön yhteistyöstä, haluaisi saada kattavan kuvan, olisi tutkijoilla oltava pääsy yrityksen sisäisiin tietokantoihin. Näiden tietojen avulla voitaisiin saada luotettava kokonaiskuva avoimen koodin kehitysmallin ja Linux-yhteistyön vaikutuksesta itse tuotetun ohjelmiston laatuun. Tämä mahdollistaisi myös yrityksen ja Linux-yhteisön ohjelmakoodin laadun vertaamisen.

Sähköpostilistoilla käydyn teknisen keskustelun merkitys projektille on hyvin vaikeasti arvioitavissa. Tämän selvittämiseksi tutkimuksessa olisi pitänyt tutustua syvällisesti käytyihin keskusteluihin ja arvioida niiden merkitys. Tarkan kuvan saamiseksi olisi tarpeellista tehdä kyselytutkimus teknisten keskustelujen aloittajille ja tiedustella heidän näkemyksiään keskustelujen hyödyllisyydestä. Joka tapauksessa kaikille avoin teknisiä kysymyksiä käsittelevä foorumi on projektille hyödyllinen työväline.

Tämä tutkimus jättää täysin avoimeksi sen, kuinka paljon ylimääräistä työtä yritys joutuu tekemään siirtyessään käyttämään Linux-kernelkehitysmallia. Avoimen kehityksen mallin oppiminen tuo mukanaan kustannuksia ja haasteita. Linuxin kernelkehityksessä käytettävä GIT-versionhallinta poikkeaa merkittävästi yleisemmin käytetyistä versionhallinnoista. Yrityksen sisäinen tuotekehitys joudutaan pahimmassa tapauksessa järjestämään kokonaan uudelleen, versionhallintaa ja tuotteen ohjelmiston rakentamisprosessia myöden. Linuxin kernelohjelmiston avoimuusvaatimukset voivat myös aiheuttaa ongelmia yrityksen patenttisalkun käytössä.

Edellä mainituista syistä johtuen aiheesta tulisi tehdä laajempi ja syvemmälle menevä tutkimus, joka analysoisi tarkemmin sekä saavutetut edut, että niiden aikaansaamat kustannukset. Sen tulisi sisältää vertailuaineistoa siitä, miten tiiviisti Linux-yhteisön kanssa yhteistyötä tekevä projekti menestyy, verrattuna perinteiseen yrityksen sisäiseen projektiin. Mahdollisuuksien mukaan analyysit pitäisi ulottaa koskemaan useampia projekteja. Lisäksi tutkimuksen tulisi tarkastella avoimen koodin vaikutusta koko

projektin aikana, sen alkuperäisestä tavoiteasettelusta aina siihen saakka, kunnes tuotteen valmistus ja tuotetuki lopetetaan. Vasta näin menetellen saataisiin kattava kuva kokonaiskustannuksista ja yhteistyön merkityksestä.

Työn yhteydessä tulisi tehdä kattava tutkimus katselmointikäytännöistä. Tässä työssä ei pystytty arviomaan sitä, kuinka moni katselmoija jätti katselmoidun koodin kokonaan kommentoimatta. Tämä voitaisiin tehdä kyselytutkimuksena, vaikkakin sellaisen järjestäminen pelkästään internetissä toimivalle yhteisölle on hankalaa. Lisäksi sellaisen kattavuutta ja luotettavuutta on hyvin vaikea arvioida.

Tutkimus voisi sisältää myös yrityksen ohjelmakehittäjiin ja projektihenkilökuntaan kohdistuvan kyselyn, jolla pyrittäisiin saamaan kuva siitä, miten niin kehittäjät kuin projektin johtokin suhtautuvat avoimen koodin kehitykseen ja yhteistyöhön yrityksen ulkopuolisten kehittäjien kanssa.

Yleisemmällä tasolla tämä tutkimus pitäisi ulottaa useampaan yritykseen, ja etsiä niitä tekijöitä, jotka auttavat maksimoimaan yhteistyöstä saadun hyödyn. Samalla voitaisiin kartoittaa niitä tekijöitä, jotka pienentävät kernelin kehitysyhteistyöstä saatavia etuja.

Edellisten lisäksi tutkimuksen tulisi käsittää vertailevaa aineistoa, jossa tutkittaisiin miten samantyyppinen projekti menestyy käyttämällä alusta alkaen jäädytettyä kernelversiota ja julkaisemalla kaiken vaadittavan koodin vasta projektin päätyttyä.

Lisäksi yrityksen on mahdollista toimia avoimen ja suljetun kehityksen välimaastossa. Tällöin yritys toimii esimerkiksi sisäisesti kuten Linuxin kernelkoodia kehittävä yhteisö.

6.5 Loppusanat

Tässä työssä tehty tutkimus on ollut monella tapaa hyödyllinen allekirjoittaneelle. Olen toiminut useissa suljetun ja avoimen koodin kerneltason ohjelmistoprojekteissa niin ohjelmistosuunnittelijana, projektipäällikkönä kuin koko kernelkehityksestä vastaavan osaston esimiehenä. Tämänkaltainen tutkimuksen tekeminen on ollut mielessäni vuosia. Tutkimuksessa löydetyt hyödyt olivat paljolti ennakko-odotusten mukaisia. Tuloksien ehkä liiankin positiivinen kuva jätti jonkin verran ajattelemisen aihetta. Nagatiivisia puolia ei ole ilmeisesti edes mahdollista löytää tällaisella, vain julkisesti saatavilla olevaan aineistoon perustuvalla tutkimuksella

Oikeiden ja tehokkaiden analyysimenetelmien valitseminen oli odotettua työläämpää. Sähköpostitietokannat eivät ole muuta kuin talletuspaikkoja sähköposteille. Postit ovat

siellä kronologisessa järjestyksessä. Kaikki muu tieto on löydettävä itse tehtävällä ohjelmalla. Toisaalta tässä käytettyjen menetelmien avulla löydetyt faktat paljastivat paljon tämän tutkimuksen rajauksen ulkopuolella olevaa tietoa. Tämä tieto koski muun muassa Linuxin kernelyhteisön verkottumisen mekanismeja ja hierarkioita.

Kaikenkaikkiaan tekemäni tutkimus synnytti suuren joukon uusia kysymyksiä. Avoimen koodin projektin hyödyt näyttävät ilmeisiltä, mutta absoluuttisen hyödyn mittaaminen on hyvin vaikea tehtävä. Jokainen projekti on erilainen, ja jokainen tuote on erilainen. Kattavaa ja yleispätevää tietoa voidaan saada vain ottamalla mukaan useita eri tavoin toteutettuja projekteja.

Lähdeluettelo

[1] Nikkanen, Tuula: Linuxin tarina. Gummerus kirjapaino Oy, Jyväskylä 2000., s. 43–90

[2] Apache web server:[online] http://httpd.apache.org/.[viitattu 13.2.2010]

[3] IBM Linux reliability research:[online] http://kb.cospa-project.org/retrieve/3087/Putting_Linux_reliability_to_the_test.png.[viitattu 3.5.2010]

[4] Howto do Linux Kernel Development:[online] http://lwn.net/Articles/160191/. [viitattu 3.5.2010]

[5] SD Cards:[online] http://www.sdcard.org/developers/tech/.[viitattu 3.5.2010]

[6] SD Technology Overview:[online] http://www.sdcard.org/developers/tech/[viitattu 17.3.2010]

[7] MMC technology:[online] http://www.mmca.org/[viitattu 6.4.2010]

[8] Motorola 68332 processor:[online] http://mpl.ch/t2211.html[viitattu 3.5.2010]

[9] Nokia N900:[online] http://maemo.nokia.com/n900/specifications/[viitattu 3.5.2010]

[10] TI3430TRM, Technical Reference Manual for OMAP3430:[online] http://www.ti.com/lit/swpu114 [viitattu 16.3.2010]

[11] OneNand technology:[online] http://www.samsung.com/global/business/semiconductor/products/fusionmemory/downloads/onenand_brochure_200609.pdf[viitattu 3.5.2010], s.4

[12] PDA Master:[online] http://pdadb.net/index.php?m=pdamaster.[viitattu 3.5.2010]

[13] Windows CE käyttöjärjestelmä:[online] http://www.microsoft.com/windowsembedded/en-us/products/windowsce/default.mspx.[viitattu 3.5.2010]

[14] GPL license:[online] http://www.gnu.org/licenses/gpl-2.0.html[viitattu 13.2.2010]

[15] Nikkanen, Tuula: Linuxin tarina. Gummerus kirjapaino Oy, Jyväskylä 2000., s. 82

[16] Kroah-Hartman, Greg; Gorbet, Jonathan; McPherson, Amanda: Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and

Who is Sponsoring It.[online][viitattu 16.2.2010] http://www.linuxfoundationorg.org/publications/whowriteslinux.pdf, Linux Foundation, 2009. , s. 5

[17] Linuxin Tiedostojärjestelmät: [online] http://www.yolinux.com/TUTORIALS/LinuxClustersAndFileSystems.html. [viitattu 5.5.2010]

[18] EXT3 File System:[online] http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html[viitattu 10.3.2010]

[19] JFFS2 Fileile System:[online] http://linux-mtd.infradead.org/~dwmw2/jffs2.pdf. [viitattu 10.3.2010]

[20] Internet protokolla versio 4:[online] http://www.ietf.org/rfc/rfc791.txt[viitattu 5.5.2010]

[21] Internet protokolla versio 6:[online] http://www.ietf.org/rfc/rfc2460.txt[viitattu 5.5.2010]

[22] Kroah-Hartman, Greg; Gorbet, Jonathan; McPherson, Amanda: Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It.[online][viitattu 16.2.2010] http://www.linuxfoundationorg.org/publications/whowriteslinux.pdf, Linux Foundation, 2009. s. 2-3

[23] GIT version control system:[online] http://git-scm.com/. [viitattu 13.2.2010]

[24] GMANE, linux mail archive,[online]http://gmane.org/export.php[viitattu 18.3.2010]

[25] Linux CodingStyle:[online] http://lxr.linux.no/#linux+v2.6.32/Documentation/CodingStyle. [viitattu 13.2.2010]

[26] LGPL-lisenssi:[online] http://www.gnu.org/licenses/lgpl.html.[viitattu 4.5.2010]

[27] MIT-lisenssi:[online] http://www.opensource.org/licenses/mit-license.php. [viitattu 4.5.2010]

[28] FreeBSD-lisenssi[online] http://www.freebsd.org/copyright/freebsd-license.html. [viitattu 4.5.2010]

[29] Apache-lisenssi[online] http://www.apache.org/licenses/[viitattu 4.5.2010]

[30] Unix-käyttöjärjestelmä:[online] http://www.unix.org/[viitattu 4.5.2010]

[31] Dahlander, Linus; Magnusson, Matts, G: Relationships between open source software companies and communities: Recearch Policy volume 34, 2005. s. 491

[32] Raymond, Eric, S: The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly Media, Revised edition 2001. s. 148

[33] Lakhani, K. R.;Wolf, R.G.:Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Project. Working paper. MITSloan School of Management/The Boston Consulting Group.http://freesoftware.mit.edu/papers/lakhaniwolf.pdf

[34] Corbet, Jonathan: Linux Kernel Report 2009.[online] http://www.osadl.org/fileadmin/dam/presentations/RTLWS11/jcorbet-kernel-report.pdf [viitattu 16.2.2009], s. 6

[35] Lee, Gwendolyn, K.; Cole, robert, E: The Linux Kernel Development As A Model of Knowledge Creation. Haas School of Business, University of California, Berkeley, 2000, s.24

[36] Henckel, Joachim; Tins, Mark: Munich/MIT Survey: Development of Embedded Linux. Institute for Innovation Research, Technology Management and Entrepreneurship, University of Munich, 2004, s. 17

[37] Lerner, J ;Tirole, J: The open source movement: Key research questions, European Economic Review 45, 2001 s. 819-826

[38] Feller, Joseph; Hissam, Scott, A; Lakhani, Karim, R: Perspectives on Free and Open Source Software. Mit Press, 2007, s. 1-45

[39] Resnick, P: RFC 2822: Internet Message Format, huhtikuu 2001. Status:proposed standard.[online] shttp://www.apps.ietf.org/rfc/rfc2822.html[Viitattu 12.02.2010].

[40] GNU Patch[online] http://mesh.dl.sourceforge.net/project/souptonuts/GNU%20Documentation/Original%20Files/GNU_diff.pdf[viitattu 6.4.2010]

[41] Arnold, J, S; Milton, J, C: Introduction to Probapility and Statistics, Third Edition. McGraw Hill 1995, s. 321

[42] Development board[online] http://beagleboard.org/[viitattu 6.4.2010]

[43] Katrina Maxwell, Luk Van Wassenhove, Soumitra Dutta: European Space, Military

and Industrial Applications, IEEE Transactions on Software Engineering, Vol. 22, No. 10, October 1996, pp. 706 -718.

[44] Boehm, B., and Basili, V.R. 2001. "Software Defect Reduction Top 10 List," Computer 34, 1 (Jan. 2001), 135-137; www.cs.umd.edu/projects/SoftEng/ESEG/papers/82.78.pdf

[45] Jones, C. "Measuring Defect Potentials and Defect Removal Efficiency," CrossTalk, June 2008; www.stsc.hill.af.mil/crosstalk/2008/06/0806jones.html

[46] Lawrence G. Votta : Does every inspection need a meeting? In Proceedings of the ACM SIGSOFT 1993 Symposium on Foundations of Software Engineering, December 1993.

[47] [online] http://smartbear.com/docs/BestPracticesForPeerCodeReview.pdf.[viitattu 28.3.2010]

Liitteet

Liite 1. Älypuhelien kellotaajuuksien ja ROM/RAM-muistin muutoksessa käytettyjen puhelien lista. Tiedot koottu PDAmasterista[12]

Tyyppi	Vuosi	MHz	RAM	ROM
Nokia 9110i Communicator	1999	33	4	4
Nokia 9210 Communicator	2001	52	8	16
Nokia 7650	2002	104	8	16
Nokia 9290 Communicator	2002	52	8	32
Nokia 9210i Communicator	2002	52	8	32
Nokia 9210c Communicator	2002	52	8	32
Nokia 3620 (Nokia Shrek)	2003	104	8	16
Nokia 6600 (Nokia Calimero)	2003	104	16	24
Nokia 3660 (Nokia Sylvester)	2003	104	8	16
Nokia N-Gage (Nokia Starship)	2003	104	16	16
Nokia 3600	2003	104	8	16
Nokia 3650 (Nokia Cameron)	2003	104	8	16
Nokia 3230	2005	123	16	24
Nokia 7710	2004	168	32	128
Nokia 6630 (Nokia Charlie)	2004	220	20	32
Nokia 6670 NAM	2004	123	16	24
Nokia 6670	2004	123	16	24
Nokia 6260	2004	123	16	24
Nokia 6620	2004	150	16	32
Nokia 7610 (Nokia Catalina)	2004	123	16	24
Nokia 7610 NAM	2004	123	16	24
Nokia N-Gage QD	2004	104	16	16
Nokia N-Gage QD NAM	2004	104	16	16
Nokia 9300/b Communicator	2006	150	64	128
Nokia 6708	2005	144	16	48
Nokia N90 (Nokia Gromit)	2005	220	32	64
Nokia N70-5	2005	220	32	48
Nokia N70	2005	220	32	48
Nokia 9300 Communicator	2005	150	64	128
Nokia 6680 (Nokia Milla)	2005	220	20	32
Nokia 6682	2005	220	20	32
Nokia 9500 Communicator	2005	150	64	128
Nokia 6681 (Nokia Cho)	2005	220	20	32
Nokia 3230	2005	123	16	24
Nokia N91 8GB (Nokia Nemo)	2006	220	64	64
Nokia N92	2006	268	64	128
Nokia E50 Metal Black	2006	235	32	160
Nokia E60	2006	220	64	128
Nokia E62	2006	235	32	160
Nokia E50-2	2006	235	32	160
Nokia 5500	2006	235	64	32
Nokia E50	2006	235	32	160
Nokia E70-2	2006	220	64	128
Nokia N73-5	2006	220	64	128

Nokia N93	2006	332	64	128
Nokia N73	2006	220	64	128
Nokia N72	2006	220	64	48
Nokia E70	2006	220	64	128
Nokia E61	2006	220	64	128
Nokia N91 (Nokia Nemo)	2006	220	64	64
Nokia N71	2006	220	64	64
Nokia N80	2006	220	64	128
Nokia 3250 (Nokia Thunder)	2006	220	48	32
Nokia 9300i Communicator	2006	150	64	128
Nokia 9300/b Communicator	2006	150	64	128
Nokia N82	2007	332	128	256
Nokia 6120c-5 classic	2007	369	64	128
Nokia E51	2007	369	96	256
Nokia N81	2007	369	96	128
Nokia N81 8GB	2007	369	96	7630
Nokia N95 8GB	2007	332	128	7630
Nokia N95-3 NAM	2007	332	128	256
Nokia 5700 XpressMusic	2007	369	64	128
Nokia 6120 classic	2007	369	64	128
Nokia 6121 classic	2007	369	64	128
Nokia E61i	2007	220	64	128
Nokia E65	2007	220	64	128
Nokia 6110 Navigator	2007	369	64	128
Nokia E90 Communicator	2007	332	128	256
Nokia N77	2007	220	64	128
Nokia N75	2007	220	64	128
Nokia N76	2007	369	96	128
Nokia N95	2007	332	64	256
Nokia N93i	2007	332	64	128
Nokia N93i-5	2007	332	64	128
Nokia 6290	2007	369	64	128
Nokia N79-3	2009	369	128	256
Nokia N79 Eco	2009	369	128	256
Nokia N79-2	2009	369	128	256
Nokia N85-2 NAM	2008	369	128	256
Nokia N96c	2008	264	128	15258
Nokia E63	2008	369	128	256
Nokia 6650-2 fold	2008	369	64	128
Nokia N96-3 NAM	2008	264	128	15258
Nokia 5800 / 5800d-1 XpressMusic (Nokia Tube)	2008	369	128	256
Nokia N79	2008	369	128	256
Nokia N85	2008	369	128	256
Nokia E66-3 (Nokia Dora)	2008	369	128	256
Nokia N96	2008	264	128	15258
Nokia E66-2 (Nokia Dora)	2008	369	128	256
Nokia E71-3	2008	369	128	256

Nokia 6210 Navigator	2008	369	64	256
Nokia 5320 XpressMusic	2008	369	128	256
Nokia E71-2	2008	369	128	256
Nokia 6650 fold	2008	369	64	128
Nokia 6220 classic	2008	369	128	256
Nokia E71	2008	369	128	256
Nokia E66 (Nokia Dora)	2008	369	128	256
Nokia N78	2008	369	96	256
Nokia 6124 classic	2008	369	64	128
Nokia 6122c	2008	369	64	128
Nokia N95 8GB NAM	2008	332	128	7630
Nokia E51-2	2008	369	96	256
Nokia N97 Mini	2009	434	128	7630
Nokia 5800 NAM Navigation Edition (Nokia Tube)	2009	369	128	256
Nokia E72-2 NAM	2009	600	128	512
Nokia X6 / X6-00 32GB (Nokia Alvin)	2009	600	128	31260
Nokia N900 (Nokia Rover)	2009	600	256	30518
Nokia E72	2009	600	128	512
Nokia 6720-2 classic	2009	600	128	128
Nokia 6760 slide	2009	600	128	256
Nokia 5230	2009	434	128	256
Nokia 6790 slide	2009	434	128	256
Nokia 5530 XpressMusic	2009	434	128	256
Nokia N86-2 8MP	2009	434	128	7630
Nokia 5800 Navigation Edition (Nokia Tube)	2009	369	128	256
Nokia 6790 Surge	2009	369	128	256
Nokia 6710 Navigator	2009	600	128	256
Nokia 5730 XpressMusic	2009	369	128	256
Nokia E52	2009	600	128	256
Nokia 5730c XpressMusic	2009	600	128	256
Nokia N97c	2009	434	128	30518
Nokia E55	2009	600	128	256
Nokia 6720 classic	2009	600	128	128
Nokia 6730 classic	2009	600	128	128
Nokia N97	2009	434	128	30518
Nokia N86 8MP	2009	434	128	7630
Nokia N97-2	2009	434	128	30518
Nokia 5630 XpressMusic	2009	600	128	256
Nokia E75-2	2009	369	128	256
Nokia E71x	2009	369	128	256
Nokia E75	2009	369	128	256
Nokia E63-3	2009	369	128	256
Nokia N79 Active	2009	369	128	256
Nokia 5800 NAM / 5800d-1b XpressMusic (Nokia Tube)	2009	369	128	256
Nokia N85-3	2009	369	128	256
Nokia E63-2 NAM	2009	369	128	256
Nokia N79-3	2009	369	128	256

Nokia N79 Eco	2009	369	128	256
Nokia N79-2	2009	369	128	256
Samsung SGH-D710	2004	192	32	64
Samsung SCH-i830 / IP-830w	2006	520	64	128
Samsung SGH-i300	2005	416	64	32
Samsung SCH-M600	2005	520	64	64
Samsung SGH-D730	2005	192	32	64
Samsung SCH-i730	2005	520	64	128
Samsung SGH-D720	2005	192	32	64
Samsung SGH-i607 BlackJack	2006	220	64	128
Samsung SGH-i320n	2006	416	64	128
Samsung SGH-i310	2006	416	64	128
Samsung SGH-i320	2006	416	64	128
Samsung SGH-Z600	2006	192	64	128
Samsung SGH-i300x	2006	416	64	32
Samsung SGH-i750	2006	416	64	128
Samsung SCH-i830 / IP-830w	2006	520	64	128
Samsung SGH-i780	2008	624	128	256
Samsung SGH-i617 Jack	2008	260	128	256
Samsung SGH-i608	2008	220	48	128
Samsung SGH-i450	2008	330	64	128
Samsung SPH-M4650 Multi-Touch	2007	624	128	256
Samsung SPH-M8200	2007	520	64	384
Samsung SGH-i617 BlackJack II	2007	260	128	256
Samsung SCH-i760	2007	400	64	128
Samsung SGH-i620	2007	416	64	128
Samsung SGH-i760	2007	520	64	128
Samsung SGH-i400	2007	200	64	128
Samsung SGH-i710	2007	416	64	128
Samsung SGH-i718	2007	416	64	128
Samsung SPH-M6200 Ultra Messaging	2007	220	64	128
Samsung SCH-M620	2007	220	64	128
Samsung SPH-M8100	2007	520	128	256
Samsung SGH-i520 / SGH-i520v	2007	330	64	128
Samsung SGH-i600 HSDPA	2007	220	64	128
Samsung SGH-i777 Saga	2008	400	128	256
Samsung GT-i8510L 8GB	2008	330	128	7630
Samsung GT-i8510C INNOV8	2008	330	128	7630
Samsung GT-C6620	2008	330	128	128
Samsung GT-i7110	2008	330	128	256
Samsung SCH-i910 Omnia 8GB	2008	330	128	7680
Samsung SPH-M4800 Ultra Messaging II	2008	624	128	256
Samsung SCH-M490 T*OMNIA	2008	806	128	7680
Samsung SCH-M495 T*OMNIA	2008	806	128	15360
Samsung SGH-i617T	2008	260	128	256
Samsung SCH-i770 Saga	2008	400	128	256

Samsung SCH-M480	2008	400	128	256
Samsung GT-i8510 INNOV8 16GB	2008	330	128	15258
Samsung SGH-i900L	2008	624	128	7680
Samsung GT-i8510 INNOV8 8GB	2008	330	128	7630
Samsung SGH-i907 Epix	2008	624	128	256
Samsung SGH-i740C	2008	624	64	256
Samsung SGH-i908E	2008	624	128	15360
Samsung SPH-M4655 Multi-Touch II	2008	624	128	256
Samsung SGH-i900 / SGH-i908 Omnia 16GB	2008	624	128	15360
Samsung SGH-i900 / SGH-i900V Omnia 8GB	2008	624	128	7680
Samsung SGH-i740	2008	624	64	256
Samsung SGH-L870	2008	264	128	256
Samsung SGH-i550w	2008	330	128	256
Samsung SGH-i550	2008	330	128	256
Samsung SGH-i200	2008	264	64	128
Samsung SGH-i728	2008	624	128	256
Samsung SGH-i788	2008	624	128	256
Samsung SGH-i560 / SGH-i560v	2008	330	128	256
Samsung SCH-M470	2008	330	128	256
Samsung SGH-i616	2008	260	128	256
Samsung SGH-i326 Black	2008	416	64	128
Samsung SGH-G810	2008	330	128	256
Samsung SGH-i640 / SGH-i640v	2008	416	64	128
Samsung SPH-i325 ACE	2008	416	64	128
Samsung SGH-i780	2008	624	128	256
Samsung SGH-i617 Jack	2008	260	128	256
Samsung SGH-i608	2008	220	48	128
Samsung SGH-i450	2008	330	64	128
Samsung SCH-M710 T*OMNIA II	2009	800	256	1950
Samsung SCH-M715 T*OMNIA II	2009	800	256	3900
Samsung SPH-M8400 Show OMNIA	2009	800	256	3814
Samsung SCH-M720 OMNIA Pop	2009	800	128	512
Samsung SCH-i920 Omnia II 8GB	2009	800	256	7680
Samsung GT-i8000L Omnia II M8 8GB	2009	800	256	7836
Samsung SPH-M7350 OZ Omnia	2009	800	256	7680
Samsung GT-i8320 Protector	2009	600	256	15360
Samsung SCH-M510	2009	600	256	15360
Samsung SGH-t939 Behold II	2009	600	320	512
Samsung GT-i5700 Galaxy Spica	2009	800	128	512
Samsung GT-B7300 OmniaLITE (Samsung Buckingham)	2009	800	128	512
Samsung GT-B7320L OmniaPRO	2009	528	256	256
Samsung GT-i6410 M1	2009	600	256	1024
Samsung SPH-M900 Moment	2009	800	288	64
Samsung SCH-i329	2009	800	128	512
Samsung GT-i8000T Omnia Icon M8 8GB	2009	800	256	7836
Samsung GT-B7610 OmniaPRO (Samsung Louvre)	2009	800	256	1908

Samsung GT-B7330 OmniaPRO	2009	528	256	512
Samsung GT-i8000U M2 2GB	2009	800	256	1908
Samsung GT-i8000L Omnia II M16 16GB	2009	800	256	15612
Samsung GT-B7320 OmniaPRO	2009	528	256	256
Samsung GT-i7500L Galaxy	2009	528	192	7630
Samsung SPH-i350 Intrepid	2009	528	256	512
Samsung GT-B7620 Giorgio Armani	2009	800	256	7630
Samsung i220 Code	2009	800	128	128
Samsung GT-i8000 Omnia II M16 16GB	2009	800	256	15672
Samsung GT-i8000 Omnia II M2 2GB	2009	800	256	1908
Samsung GT-i8000 Omnia II / GT-i8000H M8 8GB	2009	800	256	7836
Samsung GT-C6625v	2009	393	128	256
Samsung GT-i8910 16GB / Omnia HD	2009	600	256	15258
Samsung GT-i7500 Galaxy	2009	528	192	7630
Samsung SCH-M830	2009	800	128	1950
Samsung GT-i8910 HD 8GB	2009	600	256	7630
Samsung SGH-i637 Jack	2009	528	256	256
Samsung GT-B7300C (Samsung Buckingham)	2009	800	128	512
Samsung SGH-i627 Propel Pro	2009	528	128	256
Samsung GT-C6625 Valencia	2009	393	128	256
Motorola MPx200	2003	132	32	32
Motorola A920	2003	168	32	24
Motorola A760	2003	200	16	32
Motorola MPx220	2004	204	32	64
Motorola A1000	2004	168	32	48
Motorola A780	2004	312	48	64
Motorola E680 / E680g	2004	312	48	64
Motorola A768i	2004	200	32	64
Motorola A768	2004	200	32	64
Motorola A925	2004	168	32	32
Motorola i920	2006	200	32	64
Motorola M1000	2005	168	32	48
Motorola A1010	2005	168	32	48
Motorola i930	2005	200	32	64
Motorola A910 / A910i (Motorola Martinique)	2005	312	48	64
Motorola E680i	2005	312	48	64
Motorola ROKR E6 (Motorola Macau)	2006	312	48	64
Motorola ROKR E2 (Motorola Sumatra)	2006	312	48	80
Motorola Q	2006	312	64	64
Motorola HC700-G	2006	416	128	128
Motorola HC700-L	2006	624	128	128
Motorola MING A1200 / A1200i / A1200r (Motorola Hainan)	2006	312	64	64
Motorola i920	2006	200	32	64
Motorola MOTO Q9c	2007	312	64	128
Motorola MOTO Q9h Global	2007	325	96	256
Motorola MOTO Q 9h (Q9h)	2007	325	96	256

Motorola MING A1200e (Motorola Hainan)	2007	312	64	64
Motorola MOTO Q Music 9m (Motorola Nelson)	2007	312	64	128
Motorola ROKR Z6	2007	312	64	128
Motorola RAZR2 V8	2007	500	64	512
Motorola RAZR2 V8 Luxury Edition	2007	500	64	1908
Motorola RIZR Z8	2007	330	64	128
Motorola MC35	2007	416	64	128
Motorola MOTOSURF A3100 (Motorola Attila)	2009	528	128	256
Motorola MOTO VE66	2008	500	64	256
Motorola VC6096	2008	624	128	256
Motorola ROKR EM30	2008	624	64	256
Motorola ZINE ZN5	2008	500	64	512
Motorola MC75 GSM	2008	624	128	256
Motorola MC75 CDMA	2008	624	128	256
Motorola RIZR Z10	2008	330	64	128
Motorola ROKR E8	2008	312	64	1908
Motorola MOTO U9	2008	312	64	128
Motorola MC70 GSM	2008	624	128	128
Motorola MC70 CDMA	2008	624	128	128
Motorola Milestone	2009	550	256	15258
Motorola Droid / Tao A855 (Motorola Sholes)	2009	550	256	15258
Motorola CLIQ (Motorola Morrison)	2009	528	256	1908
Motorola DEXT (Motorola Morrison)	2009	528	256	1908
Motorola MC9500-K CDMA	2009	806	128	512
Motorola MC9500-K GSM	2009	806	128	512
Motorola MC5574	2009	520	128	256
Motorola FR68	2009	624	128	256
Motorola FR6000	2009	624	128	256
Motorola MOTOSURF A3100 (Motorola Attila)	2009	528	128	256

Liite 2. Käytetyt Python-ohjelmat

README for mailinglistparser v0.1
==========

Software intended for mining entire linux kernel mailing lists (or any other lists that follow the same development process).

The aim is to be able to data mine all sorts of fact and figures about the mailing listactivity and the resulting patches and git commits. In short, the software can do all sorts of counting of authors, groups of authors (by domain), separating patches from comments, separating patches that did not go to the git repository from those that did go, calculating the total amount of patches, codelines (or rather, changesetlines) and the amount of contribution divided for groups. Some support code exists for exporting the info for plotting with gnuplot. Some of this data is also easily visualizable with packages like pygraphviz, see 'graphtest.py'.

The theory of operation is based on the concept of forming threads. Two sorts of threads will be formed out of the given mailbox. Threads that are formed from patches (e.g. Sent by git-send-email) and other threads (discussion threads). The former threads are broken into individual threads so that each thread contains one patch (the first item in the thread) and the subsequent ones are (possible) comments to that patch. That way we should get the most accurate representation of the actual number of patches and the actual amount of comments they have received.

functionality divided to 3 python modules:
1. parsembox.py
2. parseutils.py
3. parsegit.py

Module (1) provides the main functionality, module (2) has some utilities that are not completely relevant but which are used by (1) nevertheless. Module (3) contains all the git support currently implemented.

The primary functions in 'parsembox' are init(), thr(), and find_all_patches() which will return domain-specific datatypes. All other functions operate on these datatypes.

The init() function just returns a mailbox object, which can be iterated to spit out one mail at a time, or it can be indexed. Message-id lookups, however, are really slow, so these have been implemented by caching a lot of info and putting the cache to a global var.

The threads reside in a single dictionary, where message-ids identify the keys (or the lead messages) of every thread. The values of these keys are lists of messages, starting with the thread "header" message. These lists contain message objects, not message-ids.
The original design decision was based on the idea that that way it is sufficient to pass the threads dictionary around and everything is self-contained within it.
There are plenty of functions which return (True,False) on some attribute regarding a certain posting (or patch). is_patch(), patch_is_new_file() take a mail object as argument and return True or False. This way, they can be used to construct some boolean logic in higher level scripts.

Some functions do simple mapping from a to b, e.g. subjects() and senders() take a list of message-ids and return the respective strings of subjects or the senders of the mails.

filter_feature() is a very useful function. It can be used to filter out a certain subset of mails according to user-defined criteria. Basically it can match a simple string to a given field in the messages, e.g. From: or

Subject: fields. More complex things can be done using filter_from_thr() function in conjunction with an anonymous function.

There are some simple functions for making comparisons between the thread data and a given git database. The sw can find out (against a list of all patch file sha1's ever committed) if a given patch also exists or has existed in the git database.

Environment variables for git are necessary if the git functionality is to be used. Namely, GIT_DIR needs to be set to point to that .git directory where the git version history resides that the user wants to mine. (e.g.: export GIT_DIR=$HOME/linux/.git)

The modules have been designed to be utilized from simple python scripts. However, the codebase also lends itself to interactive usage. A sample of an interactive session is given below:

```
$ PYTHONSTARTUP=parsembox.py python
Python 2.6.5 (r265:79063, Mar 24 2010, 21:34:09)
[GCC 4.2.1 (Apple Inc. build 5646) (dot 1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> mb=init('testi.mbox')
init(): caching...
>>> l=thr(mb)
forming threads.
total_mails=2, errorcounter=0
[]
>>> patches,plens,b,c=find_all_patches(l)
***** THREAD 1[2] ([PATCH]    Support for OMAP3 SDTI (Serial Debug
Trace Interface).)
>>> vis_thr(l)
  *Tue, 25 Mar 2008 15:47:38 +0200 Roman Tereshonkov
<roman.tereshonkov@nokia.com>
<1206452858-29570-1-git-send-email-roman.tereshonkov@nokia.com>
    .Tue, 25 Mar 2008 09:17:16 -0500 "Woodruff, Richard"
<r-woodruff2@ti.com>
<3B6D69C3A9EBCA4BA5DA60D91302742903D3BA03@dlee13.ent.ti.com>
```

 # parsembox.py, basic tools for summing up a mbox archive from a mailing list

```
# functions for parsing git logs, patches etc.

import re, os
from subprocess import *

def get_git_log(searchterm):
 cmd = 'git log -s %s' % searchterm
 data = os.popen(cmd).read(-1)
 return data

def get_patch_from_git(commit):
 cmd = 'git show %s' % commit
```

```
  data = os.popen(cmd).read(-1)
  return data

def get_blob_hash_from_abbrevs(shorthashes, allhashes, cache):
  lst =[]
  for i in shorthashes:
    a, b = i  # unpack
    full_a = get_blob_hash_from_abbrev(a, allhashes)
    full_b = get_blob_hash_from_abbrev(b, allhashes)
    try:
      lst.append( (full_a[0], full_b[0]) )
    except IndexError:
      lst.append( None )
  return lst

def get_blob_hash_from_abbrev(abbrev, allhashes):
  foo =[x[1] for x in allhashes]
  result = filter(lambda x: x[0:7] == abbrev, foo)
  return result

def get_all_blob_hashes():
  "commit_sha1, path_to_look_for ->[(blob1_sha_a, blob1_sha_b) .. (blobn_sha_a, blobn_sha_b)]"
  p1 = Popen(["git", "rev-list", "--all"], stdout=PIPE)
  p2 = Popen(["git", "diff-tree", "-r", "--stdin"], stdin=p1.stdout, stdout=PIPE)
  data = p2.communicate()[0]  # discard stderr
  #:100644 100644 72b2615600d83ab4ed2585ceb9af07bdaf62641c
953c17731e0d05b2cd2d75ace3d3a2616b1d5c5f M foo.c
  rcommit = re.compile(r"^([0-9a-f]{40})$")
  r3 = re.compile(r"^:(\d{6}).(\d{6}).([0-9a-f]{40}).([0-9a-f]{40}).([A-Z]).([a-z/A-Z0-9 ._-]+)$")
  lst =[]
  cache = {}
  for i in data.split('\n'):
    mcommit = rcommit.match(i)
    if mcommit != None:
      context = mcommit.group(1)
    m = r3.match(i)
    if m != None:
      oldh = m.group(3)
      newh = m.group(4)
      changetype = m.group(5)
      fname = m.group(6)
      lst.append( (oldh, newh, changetype, fname) )
      # also build a 7->40 sha1 expansion cache
      abbrev = oldh[0:7]
      cache[abbrev] = oldh
      abbrev = newh[0:7]
      cache[abbrev] = newh
  return lst, cache

# helper functions for parsembox.py

import re
from collections import defaultdict
```

```
def findmail(mid, mbox):
    "findmail(mid, mbox) -> mail"
    # for interactive use, digs out the mail object using a given mid
    r = None
    for i in mbox:
        if unbracket(i['message-id']) == unbracket(mid):
            r = i
            break
    return r

def emptyrto(rto):
    if rto == None or rto == '' or rto == '<>':
        return True
    else:
        return False

def unbracket(string):
    r = re.compile(r".*[<]([^<>]+)[>].*")
    if string == None:
        return None
    m = r.match(string)
    if m == None:
        return string
    else:
        return m.group(1)

def unbrackets(string):  # this is a version for multiple brackets
    r = re.compile(r".*?[<]([^<>]+)[>].*?") # use non-greedy patterns
    if string == None:
        return[]
    return r.findall(string)

def sort_counterdict(d):
    keys = d.keys()
    keys.sort(lambda x,y: 2*( d[x] > d[y])-1)
    for i in keys:
        print i, d[i]

def detect_git_thread(mid):
    r = re.compile(r"<?\d+-\d+-(\d+)-git-send-email-.*")
    # support stgit signature as well (this catches the cases where there is no stgit header)
    # 20090520064227.24734.77278.stgit@oreo.research.nokia.com
    r2 = re.compile(r"<?\d+.\d{5}.(\d{1,5}).stgit@.*")
    # some old versions of git use a different message-id format
    r1 = re.compile(r"<?\d{10}(\d{4})-git-send-email-.*")
    #[40-digit sha1].[some_code].[git].[user@domain]
    r3 = re.compile(r"<?[0-9a-f]{40}.(\d+).git.(.+)@(.+).*")
    m = r.match(mid)
    m2 = r2.match(mid)
    m3 = r3.match(mid)
    m4 = r1.match(mid)
    return m or m2 or m3 or m4
```

```
def vis_thr(thr, debug=False):
    # default sort: by date/time
    l = thr.keys()
    sortfunc = lambda x,y: 2*(date2num(thr[x][0]['date']) > date2num(thr[y][0]['date']))-1
    l.sort(sortfunc)  # in-place sort: compare the dates inside the mails
    for i in l:
        c = 0
        for j in thr[i]:  # walk through the list
            c = c+1
            print ' '*c,
            s1 = '.'
            if detect_git_thread(j['message-id']):
                s1 = '*'
            if debug==False:
                print s1+j['date'], j['from'], j['message-id'].replace('\n', '')
            else:
                print s1+j['date'], j['from'], j['subject'].replace('\n', ''), '\n', j['in-reply-to']

def quantize(data, step):
    "quantize(data, step) -> new dictionary quantized to n categories of <step> items"
    a, b = min(data.keys()), max(data.keys())
    newdict = {}
    for i in range(a, b+1, step):
        keylst = filter(lambda x: x >= i and x < i+step, data.keys())
        newdict[i] = sum(data[k] for k in keylst)
    return newdict

def flatten_histo(histo):
    l =[]
    for i in histo.keys():
        for j in range(histo[i]):
            l.append(i)
    return l

def create_histo(flatlist):
    return dict([ (x, flatlist.count(x)) for x in flatlist ] )

def histogram_to_csv(histogram, step=10, filename='out.dat', title='none'):
    "inputs: histogram(dict), outputs: csv data[strings]"
    new = quantize(histogram, step)
    out = ""
    for i in new.keys():
        s = "%d,%d\n" % (i, new[i])
        out = out + s
    return out

def histogram_to_file(histogram, step=10, filename='out.dat', title='none', xscale=None, terminal='png'):
    "histogram(dict) (optional: step, filename, title, xscale, terminal) -> no outputs (side effect: out.dat,
out.script)"
    f = open(filename, 'w')
    f2 = open(filename+'.script', 'w')
    new = quantize(histogram, step)
```

```
    out=""
    for i in new.keys():
        s = "%d %d\n" % (i, new[i])
        out = out+s
    f.write(out)
    # produce a gnuplot script
    f2.write(('set terminal %s\n'%terminal))
    f2.write(('set style line 1 lt 1 lw %d\n' % (step/2)))
    if xscale == None:
        f2.write(('plot "%s" using ($1):($2) with impulses ls 1 title "%s"\n' % (filename, title)))
    else:
        f2.write(('plot[%d:%d] "%s" using ($1):($2) with impulses ls 1 title "%s"\n' % (xscale[0], xscale[1],
filename, title)))
    f.close()
    f2.close()

def cdf_to_file(histogram, step=10):
    "inputs: histogram(dict), no outputs (side effect: out.dat, out.script)"
    f = open('cdf.dat', 'w')
    f2 = open('cdf.script', 'w')
    new = quantize(histogram, step)
    out=""
    cum=0
    for i in new.keys():
        cum = cum + new[i]
        s = "%d %d\n" % (i, cum)
        out = out+s
    f.write(out)
    f2.write('set terminal png\n')
    f2.write(('set style line 1 lt 1 lw %d\n' % (step/2)))
    f2.write('plot[0:500] "cdf.dat" using ($1):($2) with impulses ls 1 title "CDF of n(codelines)"\n')
    #f2.write('plot "out.dat" using ($1):($2) with impulses\n')
    f.close()
    f2.close()

def show_histo(histogram, step=10, xrange=None, yrange=None):
    # for interactive usage, runs gnuplot on the given data
    import os
    histogram_to_file(histogram, step)
    f = open('out.script','w')
    if xrange != None:
        f.write('plot[%d:%d] "out.dat" using ($1):($2) with impulses title "foo"\n' % xrange)
    else:
        f.write('plot "out.dat" using ($1):($2) with impulses title "foo"\n')
    f.close()
    os.system('gnuplot out.script')

def get_domain(emailstr):
    m = re.compile("<?[^@]+@([^ ]+).*").search(unbracket(emailstr))
    if m != None:
        return m.group(1)

def count_by_domain(histo):
```

```
    "{contributor1:npatches,..} -> {domain1:npatches1,..domainN:npatchesN}"
    d = defaultdict(int)
    for k in histo.keys(): # go through every author
        x = get_domain(unbracket(k)) # works for full From: field and stripped addr as well
        domain = x.lower() # we want lowercase
        d[domain] += histo[k]
    return d

def get_patchseries(mid, threads):
    "find the mid of patchseries index page (0)"
    head_mid = trace_ancestry(mboxi, rto,[])
    lst =[]
    for k in threads.keys():
        if trace_ancestry(mboxi, k,[]) == head_mid:
            lst.append(threads[k])
    return lst

def count_list(lst):
    "[item1..itemn] -> dict(items, counts)"
    d = defaultdict(int)
    for i in lst:
        d[i] += 1
    return d

def count_groups(data):
    "data{} -> lengths{} (return a dict that contains counts for the keys)"
    a =[(x, len(data[x])) for x in data.keys()]
    return dict(a)

def form_contributors(threads, patchlist):
    contributors = defaultdict(int)
    for i in threads.keys():
        lst = threads[i]
        mid = lst[0]['message-id']  # thread leader
        if unbracket(mid) in patchlist:
            foo = lst[0]['from']
            contributors[foo] += 1
    return contributors

def contributions_by_domain(threads, patchlist):
    d = defaultdict(int)
    for i in threads.keys():
        lst = threads[i]
        mid = lst[0]['message-id']
        if unbracket(mid) in patchlist:
            domain = get_domain(lst[0]['from'])
            d[domain] += 1
    return d

def comments_by_domain(threads, patchlist):
    d = defaultdict(int)
    for i in threads.keys():
        lst = threads[i]
```

```
        mid = lst[0]['message-id']
        if unbracket(mid) in patchlist:
            for comment in lst[1:]:  # skip the 1st one, the patch
                domain = get_domain(comment['from'])
                d[domain] += 1
    return d

def form_commentators(threads, patchlist):
    commentators = defaultdict(int)
    for i in threads.keys():
        lst = threads[i]
        mid = lst[0]['message-id']  # thread leader
        if unbracket(mid) in patchlist:
            for comment in lst[1:]:  # skip the 1st one, the patch
                foo = comment['from']
                commentators[foo] += 1
    return commentators

def map_contributors(contriblist):
    lst =[ (unbracket(x), x) for x in contriblist ]
    return dict(lst)

def top_contributors(d, topten=10):
    "{contributors} ->[ (contributor1, n_contribs)..(contributorN, n_contribsN) ]"
    l = d.keys()
    sortfunc = lambda x,y: 2*(d[x] > d[y])-1
    l.sort(sortfunc)  # in-place sort against the values
    return dict([ (x, d[x]) for x in l[-topten:]] )

def date2num(s):
    # take something of form "Mon, 01 Jan 2009" and return the swedish notation
    # accept also the rarely appearing form without the leading day name
    m_mapping =
{'Jan':1,'Feb':2,'Mar':3,'Apr':4,'May':5,'Jun':6,'Jul':7,'Aug':8,'Sep':9,'Oct':10,'Nov':11,'Dec':12}
    r = re.compile(r"([a-zA-Z]+, )?(\d{1,2}) +([a-zA-Z]{3}) +(\d{4}) +(\d\d?):(\d\d):(\d\d).*")
    m = r.match(s)
    if m == None:  # non-match handling
        return None
    day = int(m.group(2))
    month = m_mapping[m.group(3)] # map names to numbers
    year = int(m.group(4))
    h = int(m.group(5))
    mi = int(m.group(6))
    s = int(m.group(7))
    return "%04d%02d%02d%02d%02d%02d" % (year, month, day, h, mi, s)

def datenum_normalized(s, d):
    "datenum_normalized('200102031234 +5678', (y,m,d) -> normalized_date_num"
    # a function that takes the output from date2num and normalizes it to a count of days since a given date
    mdays =[0,31,28,31,30,31,30,31,31,30,31,30,31]
    firstdate = d[0]*365+( reduce(lambda x,y: x+y, mdays[0:d[1]]) )+d[2]  # y,m,d
    dates = defaultdict(int)
    year,month,day = int(s[0:4]), int(s[4:6]), int(s[6:8])
```

```
    # calculate month2 as a cumulative sum of items in the dict
    if (year % 4)==0:  # handle some (not all) irregularities
       mdays =[0,31,29,31,30,31,30,31,31,30,31,30,31]
    month2 = reduce(lambda x,y: x+y, mdays[0:month])
    dnum = year*365+month2+day - firstdate  # normalize the date
    return dnum

def print_mailheader(m):
    print m['message-id']
    foo = date2num(m['date'])
    s = m['subject'].replace('\n','')
    print '(%s) %s\n%s writes about %s' % (foo, m['lines'], m['from'], s)

def pick_median(lst):
    lst2 = lst[:]  # make a copy
    lst2.sort()  # in-place sort
    return lst2[ len(lst)/2 ]

def summary(s_):  # a
 R-style summary function
 S = s_[:]  # make a copy
 S.sort()  # in-place sort
 min_ = min(S)
 max_ = max(S)
 avg = 1.0*sum(S) / len(S)
 med = S[ len(S)/2 ]
 q1 = S[ 1*len(S)/4 ]
 q3 = S[ 3*len(S)/4 ]
 print 'Min\t\t1st Q\t\tMedian\t\tMean\t\t3rd Q\t\tMax'
 print '%.2f\t\t%.2f\t\t%.2f\t\t%.2f\t\t%.2f\t\t%.2f' % (min_, q1, med, avg, q3, max_)
 return (min_, q1, med, avg, q3, max_)

def random_stats(S, thr, plens):
 totreps, totrows = 0,0
 tmplst, tmplst2=[],[]
 for i in S:  # calculate avgs
 nreplies, nrows = len( thr[i] ), plens[i]
 totreps += nreplies
 totrows += nrows
 tmplst.append( nreplies )
 tmplst2.append( nrows )
 median_reps = pick_median(tmplst)  # calculate medians
 median_rows = pick_median(tmplst2)
 s2 = "stats: the median patch in this set has %.2f responses and the median patch size is %d lines" %
( median_reps, median_rows )
 s1 = "stats: this set has received in average %.2f responses and the avg patch size is %d lines" %
( (1.0*totreps / len(S)), totrows/len(S) )
 return s1+'\n'+s2

def form_comments(lista, threads):
 "[mid1..midn], threads ->[comment1..commentn]"
 l =[]
 for i in lista:
```

```
        t = threads[i]  # pick thread by the id
        for j in t[1:]:  # skip first one, this is not a comment but a patch
            l.append(j)  # append the ids of all comments
    return l

def form_commented(lista, threads):
    "[mid1..midn], threads ->[commented] forms a list of mails that have received comments"
    l =[]
    for i in lista:
        t = threads[i]  # pick thread by the id
        if len(t) > 1:
            l.append(i)  # append the id of the root mail
    return l

# parsembox.py, basic tools for summing up a mbox archive from a mailing list

import os, re, email, mailbox, base64
from collections import defaultdict
from parseutils import *
from parsegit import *

# description of used data structures:
# thread_structure = dictionary where
#   keys=the id of leading or opening mail
#   values=list of emails (not ids) pertaining to that thread

# global variables
errorcounter=0
debug=0
cache=None
mbox=None
patchcache=None
commitcache=None


def init(altname="mbox.data", append=False):
    mbox = mailbox.mbox(altname)
    globals()['mbox'] = mbox  # put the mbox into global space
    # build cache...
    if append:
        cache = globals()['cache']
    else:
        cache = {}
    mid2idx = {}
    for idx in range(len(mbox)):
        i = mbox[idx]
        mid = i['message-id']
        mid2idx[unbracket(mid)] = idx
        if mid == None:
            globals()['errorcounter'] += 1
            continue  # refuse mails without a valid message-id
        cache[mid] = {} # init as a dictionary
        cache[mid]['subject'] = i['subject']
```

```
        cache[mid]['from'] = i['from']
        cache[mid]['date'] = i['date']
        cache[mid]['in-reply-to'] = i['in-reply-to']
        cache[mid]['list-id'] = i['list-id']
    globals()['cache'] = cache  # put cache into global space
    globals()['mid2idx'] = mid2idx
    return mbox

def get_mail_by_mid(mid):
    cache = globals()['cache']
    return cache[ "<"+unbracket(mid)+">" ]

def detect_gitsendemail(mail):
    r = re.compile(r"<\d+-\d+-(\d+)-git-send-email-.*@.*>")
    mid = mail['message-id']
    m = r.match(mid)
    if m == None:
        return None
    else:
        return m.group(1)

def match_patchseries(mail):
    "match_patchseries(mail) -> int or None (if wasn't patchseries)"
    # this function may not match all relevant subject lines, hence provides an underestimate
    debug = bool(globals()['debug'])
    r = re.compile(r"\[patch.* (\d+)/(\d+)\] .*", re.IGNORECASE)
    s = mail['subject']
    if s == None:
        #print 'WARNING: no subject. from: %s at %s (%s)' % (mail['from'], mail['date'], mail['message-
id'])
        #print 'clearly this is a python mail library bug. will have to skip this one.'
        # upon investigation, it seems these "ghosts mails" do not really exist in the mbox file
        # hence, we do not want to increase the errorcounter to signify a discarded mail.
        #globals()['errorcounter'] += 1
        return None
    m = r.match(s)
    if m == None:
        return None  # so, this was no patchseries
    else:
        if debug:
            print 'patch has %s subunits' % m.group(2)
        return int(m.group(1))  # return the index number of the patch

def trace_ancestry(mboxi, rto, lst):
    # some problems related to deciphering in-reply-to fields:
    # if cross-posted, gmane.org seems to mangle the Message-id: so that it changes like:
    #Message-ID: <1233921146-4046-1-git-send-email-ajay.gupta@ti.com>
    #RTO=1233921146-4046-1-git-send-email-ajay.gupta-l0cyMroinI0@public.gmane.org, len(refs)=1
    # if we de-mangle it, then the handling will only depend on finding the parent post.
    # the current approach is to silently ignore the public.gmane.org part of it.
    cache = globals()['cache']
    if emptyrto(rto): return lst # when there is no RTO, terminate recursion
    # first, dig out the message that has rto as its message-id
```

```
    messageid = "<"+rto+">"
    mid = unbracket(messageid)
    corrected, haskey = get_rid_of_gmane_org(mid)  # see if we need to correct it
    if cache.has_key(messageid) or haskey==True:
        if haskey==False:
            rto = unbracket(cache[messageid]['in-reply-to']) or ""
            lst.append(unbracket(messageid))  # put the message-id to the list
        else:
            rto = unbracket(cache[corrected]['in-reply-to']) or ""
            lst.append(unbracket(corrected))  # put the corrected message-id to the list
        if debug:
            print rto, lst
        lst = trace_ancestry(mboxi, rto, lst)  # recurse into oblivion
        return lst
    else:
        return None # cache does not have this key, hence parent is unreachable.

def get_rid_of_gmane_org(midstr):
    # there exists some kind of mangling mechanism of unknown origin.
    # this code intends to find out the original message-id
    cache = globals()['cache']
    if midstr == '<>':
        print 'unexpected input'
        raise IndexError
    r1 = None
    if '$gmane$org' in midstr:  # KLUDGE #2
        r1 = re.compile(r"<(\d+[.][A-Z]{2}\d{5})__\d{5}[.]\d+\$.*?\$gmane\$org@.*>")
    if '@public.gmane.org' in midstr:  # KLUDGE #1
        r1 = re.compile(r"<(.*)-.*@.*>")
    if r1 != None:
        m = r1.match(midstr)
        if m != None:
            mid = m.group(1)
            for k in cache.keys():  # let's trace the correct mid
                if mid in k:
                    return k, True
            # if we get this far, then our exhaustive search has failed.
            return None, False
    return None, False  # any other outcome

def form_patchseries(mboxi):
    # go through all the mails, and count the patch headers (maybe it correlates to series)
    l=[]
    for i in mboxi:
        mid = i['message-id'] or ''
        subj = i['subject'] or ''
        r1 = re.compile(r"\[PATCH.* 00?/\d{1,2}.*")
        if r1.match(subj):
            l.append(mid)
    return l
    d = {}
    for i in l:
        foo = find_parent(mid)
```

```
    d[i].append(foo)


def thr(mboxi=None):
  "thr(mbox) -> dict(threads)"
  # thr() -> dictionary of the form <messageid>:[message1..messagen]
  # thread logic includes several exceptions to the default processing:
  # 1. git-send-email posting a series where mail x+1 of n points to x as its parent
  # 2. a problem in mailer where From: field somehow gets copied into In-reply-to:
  # 3. identification of automated emails: these have[APPLIED] in their subject field
  # problem 1 will be handled by making an own thread for every git-mail containing a patch
  cache = globals()['cache']
  debug = bool(globals()['debug'])
  seriescache = defaultdict(list)
  listid_dict = defaultdict(int)
  print 'forming threads.'
  if mboxi == None:
    mboxi = init()
  threads = defaultdict(list) # empty dictionary, with list factory
  print 'total_mails=%d, errorcounter=%d' % (len(mboxi), globals()['errorcounter'])
  cnt, orphans = 0, 0
  header = None
  # phase 1: scan the patch headers for later use.
  headerlist =[]
  mids =[]
  for i in mboxi:
    mid = i['message-id'] or 'null'
    mids.append(unbracket(mid))
    foo = match_patchseries(i)
    if foo == 0:
      if debug:
        print 'found a patch header', '!'*80
      header = unbracket(mid) # we use this to remind us that the parent is a header
      headerlist.append(header)
      continue
    if foo > 0:
      if debug:
        print '#%d' % foo
    detectgit = detect_git_thread(mid)
    if detectgit != None:
      if detectgit.group(1) == '1':
        seriescache[unbracket(mid)] =[unbracket(mid)]
      if debug:
        print detectgit.group(1)
        print 'phase1: MAKE_OWN'
      header = unbracket(mid)
      # match against: ^+++ b/foo/bar.c somejunk-like-timestamp$
      r3 = re.compile(r".*[+]{3} b/(.*?).*", re.DOTALL)
      has_patch = r3.match(get_mail_contents(i))
      if debug:
        print 'HASPATCH: %s'% has_patch
      if detectgit.group(1) == '1' and has_patch == None:
        headerlist.append(header) # only no.1 is the header
```

```
# phase 2: process individual mails to threads
for i in mboxi:
    if i['message-id'] == None:
        continue        # reject mails that have incomplete headers
    rto=unbracket(i['in-reply-to'])
    refs=unbrackets(i['references']) or[]
    listids = unbrackets(i['list-id'])
    for listid in listids:
        listid_dict[listid] += 1
    if debug:
        print 'MID=%s' % i['message-id']
        print '%s writes about "%s"' % (i['from'], i['subject'])
        print 'RTO=%s, len(refs)=%d' % (rto, len(refs))
    make_this_its_own_thread = False
    # kludge 1: check for an interesting way of git-send-email to link everything
    flag=False
    detectgit = detect_git_thread(i['message-id'])
    if detectgit != None:
        if debug:
            print detectgit.group(1)
            print 'MAKE_OWN'
        # this is part of a git thread, hence we form individual threads from these patches
        make_this_its_own_thread = True
    if rto in headerlist:  # our parent is a header
        if debug:
            print '######## MATCH: HEADER'
        seriescache[rto].append(unbracket(i['message-id']))
        cnt += 1
        # we put a header to the threads as well so that comments to it can be followed
        threads[unbracket(i['message-id'])].append(i)
        if unbracket(threads[unbracket(i['message-id'])][0]['message-id']) != unbracket(i['message-id']):
            print 'FATAL: 1 a thread begins with a mail with a different message-id!'
            raise ValueError
        continue # no need to trace ancestry
    if emptyrto(rto):  # usually you see <> as the empty rto
        if debug:
            print '*** this mail had an empty rto, hence it is a "root" mail'
        cnt += 1
        # put this as the 1st on thread list
        threads[unbracket(i['message-id'])].append(i)
        if unbracket(threads[unbracket(i['message-id'])][0]['message-id']) != unbracket(i['message-id']):
            print 'FATAL: 2 a thread begins with a mail with a different message-id!'

    raise ValueError
    else:
        # KLUDGE: it seems some broken client inserts From: field also into In-reply-to field???
        if unbracket(i['in-reply-to']) == unbracket(i['from']):
            if debug:
                print 'Inreplyto: same as From:?? this mailclient is BROKEN. re-setting In-reply-to: to
NULL.'
                print '*** this mail had an empty rto, hence it is a "root" mail'
            make_this_its_own_thread = True
        if make_this_its_own_thread:
```

```
            # an exception: we need to make this a thread root
            cnt += 1
            threads[unbracket(i['message-id'])].append(i) # put this as the 1st on thread list
            if unbracket(threads[unbracket(i['message-id'])][0]['message-id']) != unbracket(i['message-id']):
                print 'FATAL: 3 a thread begins with a mail with a different message-id!'
                raise ValueError
            continue  # skip to the next for item
        # we have a mail that is a reply, trace its ancestry
        ancestry_list = trace_ancestry(mboxi, rto,[])
        # some mails can come from other mboxes, but they would not exist on the main one. detect it.
        if ancestry_list == None:
            if debug:
                print 'listid: %s' % listid
                print 'DEBUG ,skipping (orphaned)'
            orphans += 1
            continue
        else:
            for tst in ancestry_list:
                if tst not in mids:
                    print '**** ERROR ancestry_list=%s' % ancestry_list
                    ancestry_list.remove(tst)  # silently remove it
        if None in ancestry_list:
            if debug:
                print 'DEBUG: this mail has ancestors but some of them are not reachable! (orphaned)
discarding thread.'
            orphans += 1
            continue
        if debug:
            print 'this mail is a response to:'
        for j in ancestry_list:
            thismail = i
            parentmail = get_mail_by_mid(j)
            if debug:
                print '    %s about "%s"' % (parentmail['from'], parentmail['subject'])
        # the last item on the list is the original parent
        if len(ancestry_list) > 0:
            original_parent = ancestry_list[-1]
            if original_parent in headerlist:
                seriescache[original_parent].append(unbracket(i['message-id']))

                if debug:
                    print 'HEADER FOUND. ', ancestry_list
                if len(ancestry_list) > 1: # if no intermediate replies, dont touch this
                    ancestry_list.remove(original_parent)
                original_parent = ancestry_list[-1]
            if debug:
                print 'DEBUg: appending this to %s' % original_parent
            if len(threads[original_parent]) == 0:
                if debug:
                    print "WTF!!!! the parent hasn't been processed yet. i'm skipping this."
                globals()['errorcounter'] += 1
                continue
            threads[original_parent].append(thismail)  # build a chain using associative list
```

```
            if len(threads[original_parent]) == 0:
                print '**** ERROR'
            if unbracket(threads[original_parent][0]['message-id']) != unbracket(original_parent):
                print 'FATAL: 4 a thread begins with a mail with a different message-id!'
                print original_parent, thismail['message-id'], len(threads[original_parent]),[x['message-id']
for x in threads[original_parent]]
                raise ValueError
        else:
            # empty ancestor list, hence the original is not reachable
            if debug:
                print 'this response is ORPHANED. (orphaned)'
                print 'listid: %s' % listid
            orphans += 1
    print cnt, len(mboxi), orphans, len(headerlist)
    return dict(threads)  # we want an immutable dict, not a defaultdict

def get_mail_contents(mail):
    "return mail contents in plaintext, whether the parts are base64-encoded or not."
    mdata = mail.get_payload()
    if type(mdata) == type([]):  # a list?
        # mail contents in several pieces. lets concat them and hope patches are in plaintext
        # go through every part and if it is base64-encoded, decode it
        text = ""
        for i in mdata:
            lst = filter(lambda x: x[0]=='Content-Transfer-Encoding', i.items())
            lst2 = filter(lambda x: x[0]=='Content-Description', i.items())
            if len(lst2) > 0:
                if lst2[0][1] == 'S/MIME Cryptographic Signature':  # a binary blob, ignore this
                    continue # skip
            if len(lst) > 0:  # key found
                if lst[0][1] == 'base64':  # assuming there's only one such key
                    contents = base64.decodestring(i.get_payload())
                else:
                    contents = i.get_payload()
                try:
                    text = text + contents
                except TypeError:
                    globals()['errorcounter'] += 1
            else:
                try: # way to circumvent multipart-inside-multipart, please FIXME
                    text = text + i.get_payload() # probably plaintext
                except TypeError:
                    globals()['errorcounter'] += 1
                    print 'WARNING: an error occurred, maybe didnt get a plaintext payload?'
    else:
        text = mdata
    return text

def get_files(mid):
    "get_files(mid) ->[ file1..fileN ] (get patched files out of a patch)"
    mb = globals()['mbox']
    m2i = globals()['mid2idx']
    lst, nlines, mdata = find_patch(mb[m2i[mid]])
```

```
    return lst

def get_hashes(mid):
    "get_hashes(mid) ->[(a_sha1, b_sha1) .. (a_sha1, b_sha1)]"
    patchcache = globals()['patchcache']
    try:
        contents = patchcache[mid]
    except KeyError:
        return None  # if not in patchcache, then this mail is not a patch.
    lst =[]
    r = re.compile(r"\nindex ([0-9a-f]{7})\.{2}([0-9a-f]{7}).?\d*\n", re.DOTALL)
    for i in r.finditer(contents):
        lst.append( (i.group(1), i.group(2)) )
    if len(lst) == 0:
        return None
    else:
        return lst

def filter_patches_for_git_inclusion(patchlist):
    lst =[]
    for i in patchlist:
        res = find_out_if_this_patch_is_in_git(get_hashes(i))
        if res:
            lst.append( i )
    return lst

def find_out_if_this_patch_is_in_git(indices):
    "indices[(a_sha1, b_sha1)..(a_sha1, b_sha1)] -> count"
    commitcache = globals()['commitcache']
    if commitcache == None:
        commitlist, abbrevcache = get_all_blob_hashes()
        globals()['commitcache'] = {'list':commitlist} # store to cache
        globals()['commitcache']['abbrev'] = abbrevcache # store
    else:
        commitlist = commitcache['list'] # fetch from cache
    if type(indices) != type([]):
        return None  # we expected a list, maybe we got None
    all = set([(x[0][:7], x[1][:7]) for x in commitlist] )  # for a list of 7 digits of the ids
    set2 = set(indices)
    if set2 <= all:  # is set2 a subset of set1?
        return True
    else:
        return False

def patch_is_new_file(mail):
    "patch_is_new_file(mail) -> True,False depending on whether the patch introduces new stuff or
amends old"
    hashes = get_hashes(mail)
    if hashes == None:  # not every patch posted contains hashes, so we have no idea
        return False
    olds =[x[0] for x in hashes]
    return bool(olds.count('0'*7) == len(olds))
```

```
def is_patch2(mail):
    "is_patch(mail) -> True, False depending on whether or not contains a patch"
    r3 = re.compile(r"^[+]{3} b/(.*?) ?.*$", re.M)
    m = r3.search(get_mail_contents(mail))
    return bool(m)

def is_patch(mid):
    "is_patch(mid) -> True, False depending on whether or not contains a patch"
    mb = globals()['mbox']
    return is_patch2(findmail(mid, mb))

def patch_has_string(mid, regex):
    mb = globals()['mbox']
    m2i = globals()['mid2idx']
    contents = get_mail_contents(mb[m2i[mid]])
    lines = contents.split('\n')
    m =[]
    for i in lines:
        m.append(regex.search(i))  # search instead of match lets us use foo instead of .*foo.*
    if m.count(None) == len(m):
        return False
    else:
        return True

def find_patch(mail):
    "find_patch(mail) ->[patchfiles], linecount, messagedata"
    # this function is based on a strict regex match, and as such only provides an underestimate
    debug = bool(globals()['debug'])
    mdata = get_mail_contents(mail)
    # match against: ^+++ b/foo/bar.c somejunk-like-timestamp$
    r3 = re.compile(r"^[+]{3} b/([a-zA-Z0-9_./-]*) ?.*$", re.M)
    lst =[]
    firstpos = 0
    for i in r3.finditer(mdata):
        if firstpos == 0:
            firstpos = i.start()
        lst.append(i.group(1))
    r4 = re.compile(r"\n-{2} ?\n\d[.]\d", re.DOTALL)
    m = r4.search(mdata)
    nogitsign = False
    if m != None:
        lastpos=m.start()
    else:
        if debug:
            print 'no git version signature, assuming that the patch continues to the end.'
        lastpos=len(mdata)  # just pretend that the rest of the mail is a patch
    nlines = mdata[firstpos:lastpos].count('\n')
    return lst, nlines, mdata

def find_all_patches(threads):
    "threads(dict) -> patch_mids(list), patchlens(dict), linedistribution(dict), counters(dict)"
    debug = bool(globals()['debug'])
    patchlist =[]
```

```
    patchlengths = {}
    npatches, totlen, c = 0, 0, 0
    filecounter = defaultdict(int)
    authorcounter = defaultdict(int)
    linedistr = defaultdict(int)
    comment_authorcounter = defaultdict(int)
    patchdata={}

    allkeys = threads.keys()
    for i in allkeys:
        c += 1
        thr = threads[i]
        if debug:
            print '***** THREAD %d[%d] (%s)' % (c, len(thr), thr[0]['subject'])
        j = thr[0] # study only the originator of the thread
        filelist, length, mdata = find_patch(j)
        if len(filelist) > 0:
            if debug:
                print filelist, length
            foo = unbracket(thr[0]['message-id'])
            if debug:
                print foo, i
            if foo != i:
                print 'FATAL: message-id of the thread head differs from the one in threads.'
                print i, threads[i], threads[i][0]
                raise IndexError
            patchlist.append(foo)
            patchlengths[foo] = length
            patchdata[foo] = mdata
            if len(foo) < 1:
                print 'problem: no message-id?? The mail looks like:'
                print j
                continue
            linedistr[length] += 1
            totlen += length
            npatches += 1
            for j in filelist:
                filecounter[j] += 1
                author = thr[0]['from']
                authorcounter[author] += 1 # amount of files patched, not patches per se
    if debug:
        print 'total lines %d, num_patches %d out of %d threads.' % (totlen, npatches, len(allkeys))
    counters = {}
    counters['file'] = filecounter  # FIXME: filecounter currently broken
    counters['author'] = authorcounter
    globals()['patchcache'] = patchdata
    return patchlist, patchlengths, linedistr, counters

def filter_out_dupes(threads):
    # go through every mail, build up a dict of patch sha1's and find out resends and other dupes
    # then, filter the threads so that only one version of a resent patch thread survives
    # note that this provides an underestimate on the real number of resent patches since any
    # (even small) modification will cause the patch to be considered non-dupe!
```

```
    # hence, essentially this code catches RESENDs, but not comment-modify-repost flows unless
    # the patches appear in patchseries, where the modified patches in the patchseries are not
    # detected.
    debug = bool(globals()['debug'])
    d = defaultdict(list)
    newthreads = defaultdict(list)
    print len(threads)
    for mid in threads.keys():
        t = threads[mid]
        foo = get_hashes(mid)
        if foo == None:  # no hashes, handle it as a non-dupe
            if debug:
                print 'no hashes'
            newthreads[mid] = t[:]  # copy the entire list
            continue
        if debug:
            print foo
        hashes = frozenset(foo)
        d[hashes].append(mid)
        if len(d[hashes]) > 1 and debug:
            print 'DUPE: ',
            print t[0]['subject']
    # now we have a dict which has all the patchsets and a list of threads which contain them
    # next task: concatenate the reply threads and place them after a single patch!
    print 'unique hashes(hashlists): %d, hashes summed up: %d' % ( len(d), sum([ len(d[x]) for x in
d.keys() ] ) )
    for k in d.keys():
        th_keys = d[k]
        th_head = threads[th_keys[0]][0]
        th = threads[th_keys[0]]
        foo =[ threads[th_keys[0]] ]
        lst=[]
        for t in th_keys:  # go through all thread-id's, append their posts here
            lst += threads[t][1:]
        if debug:
            print lst
        thiskey = unbracket(th_keys[0])  # th_keys[0] is the 1st patch sent
        newthreads[thiskey] =[ th_head ] + lst
    return dict(newthreads)

def subjects(lst):
    return[get_mail_by_mid(x)['subject'] for x in lst]
def senders(lst):
    return[get_mail_by_mid(x)['from'] for x in lst]
def mailinglists(lst):
    return[get_mail_by_mid(x)['list-id'] for x in lst]

def filter_out_noreplies(threads, patchlist):
    lst =[]
    for mid in threads.keys():
        i = threads[mid]
        if unbracket(mid) in patchlist:
            print '%s: this mail contains a patch. thread len=%d' % (mid, len(i))
```

```
            if len(i) < 2:
                lst.append(i[0])
    # 2nd pass: filter out duplicate patches
    indices = defaultdict(int)
    for k in lst:
        idxlist = get_hashes(threads[k])
        for i in idxlist:
            if indices.has_key(i):
                print 'DUPE found!'
            else:
                indices[i] = 1
    return lst

def filter_feature(threads, subset, field, str):
    "foo(threads,subset,field,str)->[keys] (filtering out threads using a subset and a field text filter)"
    l =[]
    keys = threads.keys()
    if subset != None:
        keys = filter(lambda x: x in subset, keys)
    for k in keys:
        mail = get_mail_by_mid(k)
        if str in mail[field]:
            l.append(k)
    return l

# count_from_thr() can be implemented using len(filter_from_thr())
def filter_from_thr(lst, thr, func):
    "lst, threads, func() ->[] (filter items on lst from threads where func() returns True)"
    l =[]
    for i in lst:
        try:
            if func(thr[i]):
                l.append(i)
        except KeyError:
            print 'keyerror occurred, check the contents of the list.'
    return l

def histo_from_comments(threads):
    d = defaultdict(int)
    for k in threads.keys():
        nreplies = len(threads[k])-1
        d[nreplies] += 1
    return d

def stats(data):
    flat = flatten_histo(data)
    print 'total lines of all patches: %d' % sum(flat)
    print 'average lines: %d' % (sum(flat) / len(flat))
    print 'median patch size (lines): %d' % (flat[ len(flat)/2 ])
    print 'ERRORS: errorcounter = %d' % globals()['errorcounter']
```

```
# Pyhton-ohjelmat joilla tehdään työssä olevat histogrammit

#!/usr/bin/env python
import parsembox as m
import os, re
from collections import defaultdict

mb = m.init()
threads = m.thr(mb)
p,plens,b,c=m.find_all_patches(threads)
print 'TOTAL mails processed: %d' % len(mb)
m.stats(b)

# duplikaattien poisto
foo = m.filter_out_dupes(threads)
threads = foo

domainit =[ 'nokia.com', 'ti.com', '.']
setit = {}
for domaini in domainit:  # kerataan etsien domainia from-kentasta
 setit[domaini] = m.filter_feature(threads, p, 'from', domaini)

for domaini in domainit:
 S = setit[domaini]
 print 'SET: %s (n=%d)' % (domaini, len(S))
 print m.random_stats(S, threads, plens)
 d = defaultdict(int)
 for i in S:
   t = threads[i]
   plen = plens[m.unbracket(t[0]['message-id'])]
   d[plen] += 1
 fname = domaini
 if fname == '.': fname = 'all'
 # sitten plotataan
 sc = {True:500,False:4000}[fname!='all']
 m.histogram_to_file(d, step=10, filename=fname, xscale=(0,sc),
title='distribution of patchlines', terminal='emf')
 os.system(('gnuplot %s.script >%s.emf' % (fname, fname)))
 m.histogram_to_file(d, step=10, filename=fname, xscale=(0,sc),
title='distribution of patchlines')
 os.system(('gnuplot %s.script >%s.png' % (fname, fname)))

# viela yksi case: muut patchit paitsi nok ja ti
S1 = setit['.']
S2 = setit['nokia.com']
S3 = setit['ti.com']
S = list( set(S1) - set(S2) - set(S3) )  # settien erotus
d = defaultdict(int)
for i in S:
 t = threads[i]
 plen = plens[m.unbracket(t[0]['message-id'])]
 d[plen] += 1
fname = 'NOTnokiaNORti'
```

```
m.histogram_to_file(d, step=10, filename=fname, xscale=(0,500),
title='distribution of patchlines', terminal='emf')
os.system(('gnuplot %s.script >%s.emf' % (fname, fname))) #
openoffice syo emf-formaattia
m.histogram_to_file(d, step=10, filename=fname, xscale=(0,500),
title='distribution of patchlines')
os.system(('gnuplot %s.script >%s.png' % (fname, fname)))
```

Liite 3. Sähköpostiarkistosta sähköposteja hakeva Python-ohjelma.

```
#!/usr/bin/env python
#
# a script for fetching a certain date range from a given mail archive
#
# arguments: date1, date2 (format: yyyymmddHHMMSS)
#
# recursive logic finds out the indices of the mails after and before
# a date range. all this assumes that the set of mails is an ordered set
# (based on the date).
# this method reduces significantly the necessary amount of mails that need
# to be downloaded, for a range of N mails the amount of downloaded mails
# is approx 2 * log2(N) plus the amount of mails falling between the dates

import sys
import urllib
import email
from parseutils import *

db_addr = 'download.gmane.org/gmane.linux.ports.arm.omap'
outfile = 'omap.data'

def fetchmails(addr, begin, end):
 u = 'http://%s/%d/%d' % (addr, begin, end)
 out = ""  # empty string, for concatenation
 f = urllib.urlopen(u)
 data = "foo"  # python does not have a do..while construct
 while len(data) > 0:
   data = f.readline()
   out = out + data
 urllib.urlcleanup()
 return out

# iterative method to find out the exact mail where the True/False
state is divided
def itermails(addr, low, high, func, cmp):
 if high-low <= 1:
   return low, high  # condition for finding the end state
 print low,high
 halfway = low + (high-low)/2
 data = fetchmails(addr, halfway, halfway+1)
 m = email.message_from_string(data)
 try:
   date = int(date2num(m['date']))
 except TypeError:  # what if the date field is missing, e.g. no such mail
   date = 22222222222222  # probably out of range, so make it a "too high" date
 if bool(func(date)) ^ bool(cmp==-1):  # we know all "higher" match,
so we iterate the "low" side of the data
    newlow, newhigh = itermails(addr, low, halfway, func, cmp)
 else:
    newlow, newhigh = itermails(addr, halfway, high, func, cmp)
 return newlow, newhigh
```

```
# phase 1: do some iterations trying to find out the range of the mails
# we have no idea what is the max index of mails, so we use a
ridiculous starting value
start = 2000000
startdate = int(sys.argv[1])
enddate = int(sys.argv[2])
startidx, foo = itermails(db_addr, 1, start, lambda x: (x > startdate), 1)
endidx, foo = itermails(db_addr, 1, start, lambda x: (x <= enddate), -1)

# phase 2: now we have the range of indices, so let's download them
print 'fetching mails ranging from %d to %d and dumping everything
into %s' % (startidx,endidx,outfile)
outf = open(outfile, 'w')
for i in range(startidx, endidx, 1000):  # use 1000-mail pieces to
avoid hitting server cpu limits
 if endidx - i > 1000:
   piece = 1000
 else:
   piece = endidx - i
 data = fetchmails(db_addr, i, i+piece)
 outf.write(data)
outf.close()
```